POLITECHNIKA WARSZAWSKA

DZIEDZINA NAUK INŻYNIERYJNO-TECHNICZNYCH
INFORMATYKA TECHNICZNA I TELEKOMUNIKACJA

Rozprawa doktorska

mgr inż. Bartosz Wojciech Dobrzyński

Wielowymiarowa eksploracja repozytoriów programowych w zakresie raportów zgłoszeń oraz ich obsługi

Promotor
prof. dr hab. inż. Janusz Sosnowski

WARSZAWA 2023

Pragnę złożyć serdeczne podziękowania
mojemu Promotorowi – Panu Profesorowi Januszowi Sosnowskiemu
za nieocenione wsparcie i pomoc w trakcie przygotowywania rozprawy.

Spis treści

Streszczenie 7
Summary 8
1. Wprowadzenie 9
1.1. Motywacja pracy 9
1.2. Kontekst badań 11
2. Teza i cel pracy 14
3. Analiza repozytoriów 18
3.1. Analiza literatury 19
3.2. Charakterystyka repozytoriów programowych 22
3.3. Wielowymiarowa eksploracja cech repozytoriów 26
3.3.1. Charakterystyki ogólne 27
3.3.2. Analiza aktywności aktorów 37
3.3.3. Profile komentarzy zgłoszeń 42
3.3.4. Analiza korelacji atrybutów zgłoszeń 43
4. Analiza tekstowa raportów zgłoszeń 47
4.1. Analiza literatury 48
4.2. Analiza słowników 51
4.3. Schematy klasyfikacji 59
4.3.1. Algorytmy 60
4.3.2. Wyniki eksperymentów 65
4.4. Dyskusja 68
5. Analiza obsługi zgłoszeń 72
5.1. Ewaluacja obsługi zgłoszeń – poziom ogólny 72
5.2. Grafowy model obsługi zgłoszeń (IHG) 80
5.2.1. Przepływ zgłoszeń w modelu IHG 83
5.2.2. Algorytmy analizy grafu IHG 85
5.2.3. Charakterystyki czasowe stanów grafu IHG 95
5.2.4. Profile ścieżek obsługi zgłoszeń w modelu IHG 97
5.2.5. Badanie anomalii ścieżek 101
5.2.6. Analiza czasowa obsługi zgłoszeń w ścieżkach 108
5.2.7. Dyskusja 110
6. Podsumowanie 119
7. Bibliografia 124

8. Załączniki 132
8.1. Profile ścieżek obsługi zgłoszeń dla projektu Groovy 132
8.2. Grafy IHG dla projektu MongoDB 136
8.3. Zestaw atrybutów ścieżek obsługi zgłoszeń 138
8.4. Szczegółowe charakterystyki ścieżek obsługi zgłoszeń 140

Streszczenie

Praca poświęcona jest badaniom nad wielowymiarową analizą i eksploracją procesu obsługi zgłoszeń rejestrowanych w repozytoriach programowych (np. *JIRA, Bugzilla*). Zgłoszenia te dotyczą m.in. błędów, nowych funkcjonalności, poprawy wydajności. Analiza dostępnej literatury oraz praktyczne doświadczenia autora pokazały potrzebę opracowania bardziej zaawansowanych i szczegółowych modeli eksploracji zawartości repozytoriów zaadaptowanych do specyfiki tych danych. Badania przedstawione w rozprawie dzielą się na dwa obszary: i) eksploracja zawartości informacyjnej programowych repozytoriów zgłoszeń, ii) wielowymiarowa analiza obsługi zgłoszeń.

W pierwszym obszarze prac autor zdefiniował szczegółowe profile dotyczące m.in.: statystyk dostępnych danych, aktywności aktorów i ich zaangażowania, korelacji pomiędzy atrybutami zgłoszenia i modyfikacjami kodu. Istotnym aspektem analiz jest eksploracja opisów zgłoszeń, która różni się od standardowych technik *text miningu*. Opracowane zostały wyrażenia regularne wspomagające analizę słownikową opisów oraz przetwarzanie wstępne tekstów zawartych w repozytoriach. Zostały one wykorzystane w autorskim algorytmie klasyfikacji raportów zgłoszeń. Przebadano wpływ konfiguracji danych na dokładność klasyfikacji.

Drugi obszar prac skupia się na wielowymiarowej analizie procesu obsługi zgłoszeń. Szczegółowe badania prowadzone są z użyciem autorskiej koncepcji grafowego modelu obsługi zgłoszeń IHG (ang. *Issue Handling Graph*). Zaproponowane profile wydajnościowe, czasowe, strukturalne stanów i ścieżek umożliwiają ocenę efektywności procesów obsługi zgłoszeń. Rozpatrzono różne typy, zgłoszeń oraz perspektywy obserwacji. Autor opracował oryginalne algorytmy wyszukiwania anomalii procesu obsługi zgłoszeń.

Przedstawiona metodyka analizy została zweryfikowana dla reprezentatywnych projektów *open source* oraz projektu komercyjnego. W pracy potwierdzona została użyteczność opracowanych wielowymiarowych analiz. Pozwalają one na wskazanie potencjalnych kierunków optymalizacji procesu raportowania i obsługi zgłoszeń oraz umożliwiają przeprowadzenie jego oceny w różnych momentach cyklu życia oprogramowania.

Słowa kluczowe: inżynieria oprogramowania, repozytoria programowe, monitorowanie procesu obsługi zgłoszeń, eksploracja danych.

Summary

Disertation is devoted to research on multidimensional analysis and exploration of issue handing process which are registered in software repositories (e.g. JIRA, Bugzilla). These reports refers to: bugs, new features, performance improvements. Analysis of available literature and author's practical experience has showed the need of more advanced and detailed models of repository content mining adapted to specifity of this data. Research presented in the dissertation is divided into two areas: i) exploration of the information content of software reportes repositories, ii) multidimensional analysis of issue handling process.

In the first area of work, author has defined detailed profiles concerning, including: statistics of available data, actors' activity and their involvement, correlation between issue attributes and code modifications. An important aspect of the analysis is the exploration of issues descriptions, which differs from standard text mining techniques. Regular expressions have been developed to support dictionary analysis of descriptions and texts pre-processing contained in repositories. They were used in original classification algorithm for issue reports. Influence of data configuration on classification accuracy was examined.

Second area of work focuses on a multidimensional analysis of the issue handling process. Detailed research is carried out using the original concept of Issue Handling Graph (IHG) model. Proposed performance, time and structural profiles of states and paths make it possible to determine effectiveness of issue handling process. Different types of reports and observation perspectives were considered. Author has developed original algorithms for finding anomalies in the issue handling process.

Presented analysis methodology has been verified for representative open source projects and a commercial project. Thesis has confirmed the usefulness of the developed multidimensional analyses. They make it possible to indicate potential directions for optimizing the reporting and issue handling process and enable its evaluation at various points in the software life cycle.

Key words: software engineering, software repositories, monitoring of issue handling process, data mining.

1. Wprowadzenie

1.1. Motywacja pracy

Monitorowanie procesu rozwoju i utrzymania oprogramowania w efektywny sposób jest krytycznym aspektem prowadzenia projektów informatycznych. Współczesne repozytoria kodu oraz obsługi zgłoszeń zapewniają wsparcie tego procesu na poszczególnych etapach wytwarzania oprogramowania. Repozytoria kodu projektu służące do zarządzania zmianami mogą być oparte między innymi o system GIT [1] lub SVN [2]. Od roku 2010 w Google zauważalny jest wzrost liczby zapytań o system GIT, który przerósł liczbę wyszukań frazy SVN [3]. Główną zaletą GIT jest zastosowane podejście do budowy repozytorium oraz jego kompresji. System oparty jest o decentralizację, tzn. każdy programista pracuje na swojej lokalnej wersji repozytorium, gdzie zapisuje zmiany; ma także możliwość łączenia tych zmian (*mergowania*) do centralnego repozytorium. SVN natomiast opiera się o scentralizowane repozytorium, w którym programiści pracują na lokalnej kopi, a ich zmiany zapisywane są jedynie w centralnej bazie danych. GIT jest optymalniejszy również pod względem przestrzeni dyskowej. Przykładowo, projekt Mozilla zapisany w SVN zajmuje 12 GB, zaś ta sama wersja projektu zapisana w GIT potrzebuje jedynie 420 MB. Oprócz wersjonowania kodu, obydwa rozwiązania mogą zapewnić zespołowi projektowemu możliwość weryfikacji zmian przed połączeniem (ang. *code review*). Jest to jedna z wielu cech dostępnych w repozytoriach opartych o system GIT (np. *Bitbucket, GitHub, SourceFrog, CodeBase*), które znacząco wpływają na jakość wytwarzanego kodu i jednocześnie ułatwiają codzienną pracę programistów.

Obsługa zgłoszeń projektowych (zarówno nowych funkcjonalności, jak i błędów) – wspierana jest przez narzędzia klasy systemów zarządzania zgłoszeniami (ITS, ang. *Issue Tracking Systems*), takie jak: *JIRA* [4], *Bugzilla* [5], *GitHub Issues* [6], *Redmine* [7], *OTRS* [8] lub *MantisBT* [9]. Aplikacje te udostępniają możliwość zarządzenia zgłoszeniami, monitorowanie postępów prac poprzez zmianę statusów oraz kooperację i komunikację między aktorami biorącymi udział w pracach projektowych.

W swoim 12-letnim doświadczeniu komercyjnym na różnych stanowiskach (Konsultant IT, Analityk, Architekt Systemu, Lider Zespołu) zauważyłem problemy towarzyszące raportowaniu procesu wytwarzania oprogramowania oraz możliwość oceny jego jakości, dzięki eksploracji danych zawartych w repozytoriach projektowych. Po przeanalizowaniu wielu repozytoriów zgłoszeń projektowych oraz repozytoriów kodu, projektów *open source* i komercyjnych, spostrzegłem potrzebę przeprowadzenia dogłębnych

analiz (drobnoziarnistych), które znacząco różnią się od ogólnych przeglądów prezentowanych w literaturze. Szczegółowa weryfikacja danych dostępnych w repozytoriach projektowych (kodu i zgłoszeń), oraz ich wzajemne korelacje pokazują ich istotność w lepszym zrozumieniu procesów zachodzących w czasie wytwarzania oprogramowania oraz ich niedoskonałości.

Dane zawarte w repozytoriach projektowych, w dużym zakresie są opisowe i częściowo ustrukturalizowane. Zdecydowanie brakuje im formalizacji i daje dużą swobodę zarówno zarządzającym repozytoriami, jak również uczestniczącym aktorom. Stąd też klasyfikacja i ekstrakcja istniejących informacji jest wyzwaniem z punktu widzenia eksploracji danych, w szczególności analiz tekstowych, które stanowiły istotną część przeprowadzonych badań.

Dane pozyskane z repozytoriów umożliwiają wykrycie anomalii procesu oraz opracowanie usprawnień. Zgłaszane problemy często wymagają analizy przed rozwiązaniem, jednak ostatecznie może okazać się, że nie są to prawdziwe błędy. Przykładowo, zgłoszenie może wynikać z nieznajomości oprogramowania lub być duplikatem wcześniej zgłoszonego błędu. Zgłoszenia nowych funkcjonalności bądź modyfikacji istniejących, czasami generuje nowe zadania techniczne, które niezależnie od źródła ich pochodzenia, stanowią koszt dla projektu. Mianowicie, muszą zostać przypisane do osoby lub kilku osób, które przeprowadzą ich wstępną ocenę (ocenę zasadności zgłoszenia). Istotne zatem jest pozyskanie odpowiedniej wiedzy z danych dostępnych w repozytoriach projektowych i wyciągnięcie wniosków, które pozwolą na optymalizację odpowiednich procesów.

W analizie procesów wytwarzania i utrzymania oprogramowania szczególnie wartościowe są informacje zawarte w systemach monitorowania projektów (JIRA, Bugzilla). Zagadnieniu temu poświęcono wiele prac w literaturze, w tym również prowadzonych w Instytucie Informatyki Politechniki Warszawskiej, w których brałem udział [10, 11]. Szczególnie wartościowe są badania z zakresu analizy ścieżek obsługi problemów i opracowany system analizy grafów PHG (ang. *Problem Handling Graphs*), który jest systematycznie rozwijany w zakresie integracji z systemami ITS, analizy tekstowej i wyszukiwania anomalii ścieżek [12]. Ostatnio uwzględniono w nim specyfikę nowych technologii bazujących na koncepcji *SCRUM* oraz wyniki szczegółowych (drobnoziarnistych) analiz zawartości repozytoriów programowych.

Opracowany przeze mnie program *BugAnalysis* analizy grafów obsługi błędów PHG [12] był wykorzystany w prowadzonych pracach w Instytucie [10, 11, 13, 14], stał się podstawą stworzenia rozszerzonego modelu IHG (ang. *Issue Handling Graph*) wraz z oryginalnymi algorytmami analizy, które tworzą rozbudowany program *IssueAnalyzerTool*.

Uwzględniłem tu różne perspektywy obserwacji oraz szereg cech zgłoszeń (typ, priorytety, profile obsługi). Praca przedstawia autorskie profile, które znacząco wyróżniają zaprezentowane badania na tle innych publikacji. Istotną różnicą względem dostępnych badań jest wyodrębnienie różnych klas obsługiwanych zgłoszeń oraz profili operacyjnych. W literaturze skupiono się głównie na błędach, traktując je w sposób jednorodny. Opracowane metody analizy umożliwiają ocenę oraz identyfikację niedostatków raportowania zgłoszeń oraz ich obsługi. Wyniki takiej analizy są przydatne w optymalizacji tych procesów.

1.2. Kontekst badań

W literaturze można wyróżnić grupy publikacji, które skupiają się na poszczególnych tematach związanych z analizą danych lub wsparciem procesów obsługi zgłoszeń. Na szczególną uwagę zasługują prace związane z tak zwanymi modelami wzrostu niezawodności oprogramowania (*SRGMs*, ang. *Software Reliability Growth models*) [15, 16], a także na ewaluacji i usprawnieniu procesów wytwarzania i monitorowania oprogramowania [17-21]. Autorzy tych prac bazują jednak na ogólnych danych dotyczących wykrytych błędów i ich korekt. Dostępne są również publikacje [22, 23], które skupiają się na ocenie wartości informacyjnych dodawanych zgłoszeń w repozytoriach błędów. W tym wypadku zakres danych, na których prowadzone są analizy jest ograniczony. Warto również zaznaczyć, że artykuły te skupiają się na jednej perspektywie, np. na problemie duplikacji zgłoszeń, bądź modelach *SRGM*, co ogranicza wnioski i nie daje pełnego obrazu procesów.

Istnieją również prace analizujące wyniki ankiet, w których zespoły projektowe oceniały wartości informacji zawartych w zgłoszeniach błędów [24-26]. W temacie tym podjęta została również próba oceny jakości danych w błędach, na podstawie klasyfikacji wybranych cech zgłoszeń [27]. Podobną ewaluację zgłoszeń błędów przedstawia [28]. Tutaj jednak autorzy skupiają się na ogólnej liczbie rozwiązanych błędów, względem liczby zgłoszonych błędów wraz z czasem ich rozwiązania. Badania są jednak oparte o projekty typu *open source* i prowadzone na ogólnym poziomie (analizy gruboziarniste). Widoczna jest potrzeba bardziej szczegółowych analiz drobnoziarnistych, które zostały opisane w kolejnych rozdziałach pracy. Są to między innymi analizy czasowe (kwartyle Q1, Q2, Q3) przepływu zgłoszeń w różnych fazach ich przetwarzania.

Kolejnym trendem w dostępnej literaturze jest ocena możliwości wsparcia procesu przypisywania zgłoszeń (ang. *bug triaging*) poprzez stosowanie rekomendacji [23] lub wykorzystywanie metod *text miningu* do klasyfikacji i wsparcia procesu ich obsługi [29, 30]. Analiza tekstowa danych znajdujących się w repozytoriach programowych jest również

jednym z obszarów moich badań. Prace te są jednak ukierunkowane na weryfikację dostępnych informacji, statystyki słowników oraz wykorzystywanie klasyfikacji w procesie wsparcia obsługi najbardziej krytycznych błędów. Podejście to znacząco odróżnia analizy przedstawione w pracy od badań dostępnych w literaturze. Większość publikacji pomija wstępną analizę danych tekstowych lub przeprowadza ją na ogólnym poziomie, z brakiem odniesienia do większej liczby projektów (zarówno komercyjnych, jak i typu *open source*). Badacze w głównej mierze skupiają się na implementacji narzędzi oraz technik, które mogą być zastosowane w predykcji przypisania odpowiedniego zgłoszenia. Pomijana jest jednak możliwość wystąpienia trudności w zastosowaniu tego podejścia w innych projektach. Dzięki skupieniu się na analizie danych dostępnych w różnych repozytoriach oraz większym zakresie badanych projektów możliwe było przedstawienie znaczących różnic pomiędzy repozytoriami projektowymi, które między innymi obejmują styl zgłaszanych błędów. W niektórych projektach przeważają zgłoszenia z jednozdaniowym opisem błędów oraz załączonymi logami (w treści błędu). W innych projektach, zgłoszenia błędu są opisem sposobu reprodukcji błędu. Szczegółowa analiza danych pokazała również trudności przeprowadzania analiz tekstowych.

Część prac z dostępnej literatury poświęcona jest wizualizacji graficznej danych z repozytorium [31-34]. Dostępne są także publikacje, w których autorzy starają się wykorzystać wizualizację procesów zachodzących w repozytorium, w celu wsparcia poprawnego przypisania zgłoszenia [35]. Rozprawa wnosi również znaczący wkład w ten obszar badawczy poprzez opracowany oryginalny model grafów IHG oraz powiązane z nim algorytmy.

Tematyka monitorowania procesu obsługi błędów stanowi także obszar badań prowadzonych w ZOiAK w Politechnice Warszawskiej, w efekcie czego powstało wiele ciekawych artykułów, między innymi [20], które omawiają modele *SRGM*. Prowadzone są również badania z zakresu analizy logów aplikacji [36, 37]. Wspólnym mianownikiem prac jest eksploracja danych dostępnych w repozytoriach projektowych [38]. Opracowany model IHG jest rozszerzeniem analiz opartych o model PHG, w którego określaniu miałem znaczący udział [11, 14, 39]. Koncepcja tego modelu jest wynikiem mojej pracy magisterskiej [12]. Przedstawione treści wnoszą znaczący wkład w ewaluację procesów obsługi błędów. Opracowane przeze mnie nowe narzędzie *IssueAnalyzerTool* wykazało swoją użyteczność zarówno w analizie procesów zachodzących w projektach typu *open source*, jak i komercyjnych. Wyniki moich wcześniejszych badań z [12], były również wykorzystane w pracy [13]. Ponadto pewne wyniki prowadzonych badań zostały opublikowane w [10, 11, 14].

Aktualnie prowadzone badania w ZOiAK wykazały duży potencjał modeli IHG oraz potrzebę dalszych eksploracji repozytoriów projektowych, co jest celem rozprawy.

Szereg rozpatrywanych problemów w pracy wymagał głębszej analizy literatury, którą zawarto w rozdziałach poświęconych tym problemom. Wyszczególniono tam zakres dotychczasowych badań oraz kierunki rozwijania mojej działalności naukowej.

2. Teza i cel pracy

Bazując na wieloletnim doświadczeniu zawodowym (udział i kierowanie projektami IT), analizie literatury oraz wcześniejszych pracach badawczych, określiłem następującą tezę pracy:

Analiza i ocena procesu wytwarzania oprogramowania wymaga opracowania reprezentatywnych modeli oraz metod eksploracji danych z repozytoriów zgłoszeń i repozytoriów kodu. Uwzględnienie różnych poziomów ekstrakcji i agregacji informacji oraz perspektyw obserwacji, poszerza zakres przedmiotowy monitorowania projektu i ułatwia identyfikację niedoskonałości.

Przedstawiona teza powiązana jest z następującymi zadaniami badawczymi:

1) Eksploracja zawartości informacyjnej repozytoriów zgłoszeń obejmujących opracowanie:
   a) Charakterystyk strukturalnych, czasowych oraz zdefiniowanie profili różnych aspektów informacyjnych
   b) Algorytmów analiz statystycznych i tekstowych
2) Opracowanie rozszerzonego modelu grafowej obsługi zgłoszeń (IHG – *Issue Handling Graph*) oraz metod analizy zorientowanej na:
   a) Badanie profili obsługi różnych typów zgłoszeń
   b) Detekcji anomalii oraz analizy jakości repozytoriów programowych
3) Weryfikacja opracowanej metodologii na danych z repozytoriów projektów OS (*open source*) oraz komercyjnych.

Istotnym elementem pracy było opracowanie szeregu algorytmów oraz narzędzi wspomagających eksplorację danych z repozytoriów projektowych.

Sformułowanie problemu badawczego jest wynikiem moich obserwacji oraz praktycznych doświadczeń zdobytych podczas bezpośredniego zaangażowania w wielu projektach komercyjnych, na różnych poziomach decyzyjności: konsultant, programista, architekt rozwiązania oraz lider zespołu technicznego. Zaobserwowałem wiele problemów, zachodzących w trakcie obsługi zgłoszeń. Poziom skomplikowania procesów wytwarzania oprogramowania, liczba osób zaangażowanych w projekt oraz wymaganie zachowania ciągłości dostarczanych zmian uwypukla potrzebę szczegółowego zrozumienia procesów zachodzących podczas obsługi zgłoszeń – zarówno błędów, jak i nowych funkcjonalności w projektach. Dogłębne poznanie tego procesu jest istotnym aspektem optymalizacji prac, a jego poprawa może przynieść zysk zespołom projektowym (optymalizacja obsługi zgłoszeń).

Dzięki bogatemu doświadczeniu, mogłem opracować algorytmy, metryki oraz narzędzia wspierające wielowymiarową analizę, co wyróżnia moją pracę na tle innych publikacji.

Komercyjne projekty stawiają twórcom oprogramowania coraz większe wymagania dotyczące reakcji na często zmieniającą się rzeczywistość biznesową. Znacząco wpływa to na stosowane metodyki wytwarzania oprogramowania. W ostatnich latach widoczny jest wzrost stosowania podejść zwinnych (ang. *Agile*), w szczególności metodologii *SCRUM*, która pozwala w znacznym stopniu skrócić czas dostarczania nowych funkcjonalności, dzięki podejściu iteracyjnemu. Zwiększa to liczbę procesowanych zgłoszeń, które muszą być rozwiązane w krótkim czasie. Aspekty te opisane zostały w pracy na podstawie projektu komercyjnego, w którego prowadzeniu brałem czynny udział.

Każde z repozytoriów projektowych zapewnia funkcjonalność śledzenia postępu prac poprzez diagramy i/lub wykresy, które mogą prezentować pewne uproszczone dane wykorzystywane np. przez managera projektu do monitorowania pracy. W narzędziu JIRA dostępne są również dedykowane *plug-iny* – predefiniowane komponenty, które między innymi udostępniają wykresy postępu prac w sprincie (wykres spalania, ang. *burndown chart*). Funkcjonalności te skupiają się jednak na prezentacji prostych danych bez głębszej korelacji oraz weryfikują projekt w aktualnym momencie. Dzięki bezpośredniemu zaangażowaniu w wiele projektów komercyjnych oraz dogłębnej analizie repozytoriów, zaobserwowałem, że szczegółowe dane dotyczące prac projektowych są zazwyczaj rzetelnie raportowane w repozytoriach monitorujących postęp poszczególnych zadań (JIRA, Bugzilla, Redmine, OTRS, MantisBT). Istotne dane można odnaleźć również w repozytorium kodu (Bitbucket, GitHub). Dane te wymagają ekstrakcji z wykorzystaniem serwisów API.

Żadne z dostępnych rozwiązań nie udostępnia bezpośredniego połączenia z bazą danych. Kwestia ta związana jest z integralnością oraz bezpieczeństwem danych, jednak niesie to za sobą również konsekwencje związane z dostępnością danych oraz pozyskiwaniem ewentualnych asocjacji. Przykładowo, komentarze zgłoszeń udostępnione są poprzez inny interfejs API, niż dane zgłoszenia. Implementacja obarczona jest również ryzykiem związanym z zakończeniem udostępniania pewnych funkcjonalności przez system JIRA. W czasie prac badawczych nad ekstrakcją danych z repozytoriów firma Atlassian ogłosiła wyłączenie prostego uwierzytelniania żądań (ang. *requests*) API poprzez parametry, takie jak nazwa użytkownika i hasło, na rzecz bardziej bezpiecznego podejścia wykorzystującego token API generowany przez aplikację. Zmiana ta wymagała dodatkowej modyfikacji w kodzie mojego rozwiązania zapewniającego ekstrakcję danych. Widoczna jest potrzeba implementacji

mechanizmu ekstrakcji danych wraz z ich wstępną modyfikacją do zunifikowanych struktur oraz zapis tych danych w dedykowanej bazie, tak aby przeprowadzane analizy nie były zależne od dostępności połączenia z systemem zewnętrznym. Dostęp do repozytorium kodu możliwy jest poprzez dwa konektory: pierwszy, wykorzystujący system GIT, oraz drugi, opierający się na serwisach API, udostępnionych przez repozytoria zarządzania kodem (Bitbucket, GitHub). Korelacja zmian kodu ze zgłoszeniami projektowymi jest jednak problematyczna z uwagi na brak bezpośredniej konfiguracji połączenia między repozytorium kodu, a repozytorium zgłoszeń projektowych. Narzędzia udostępniają możliwość skonfigurowania takiego połączenia, jednak jest ona często pomijana w projektach bądź niemożliwa do wykonania, z uwagi na ograniczenia techniczne infrastruktury projektu. W celu zachowania „miękkiego" połączenia między repozytoriami, programiści wykorzystują funkcjonalność komentarzy w repozytoriach zgłoszeń. Przykładowo, w projekcie komercyjnym każda zmiana w kodzie aplikacji związana ze zgłoszeniem JIRA jest dodawana jako komentarz, z odnośnikiem do wniosku zmian w repozytorium kodu (PR, ang. *pull request*). Modyfikacja kodu (ang. *commit*) powinna zawierać również identyfikator (ID) zgłoszenia JIRA w komentarzu (ang. *commit message*). Obydwie operacje są jednak obarczone błędem ludzkim, tzn. programista może zapomnieć o dodaniu komentarza z odnośnikiem do PR bądź nie umieści JIRA ID w komentarzu zmiany (ang. *commit message)*. Kwestia zachowania połączenia pomiędzy zmianami w kodzie, a zgłoszeniami jest problematyczna także w projektach typu *open source*.

Wyekstrahowane dane wymagają opracowania szczegółowych analiz łączących w sobie ocenę wartości informacyjnej poszczególnych pól, jak również analizy tekstowej i zmienności poszczególnych wartości w czasie trwania projektu. Podstawowe zestawy oceny postępu projektu dostępne są w repozytoriach, jednak, jak wspomniałem wcześniej, są one jedynie gruboziarniste i nie dają wglądu w szczegóły procesu. W pracy przedstawiłem modele szczegółowej analizy procesu obsługi zgłoszeń oraz jakości prowadzonych prac wraz z ich korelacją z repozytorium kodu. Zaprezentowana holistyczna i wielowymiarowa analiza jest istotną cechą rozprawy. Celem przeprowadzanych analiz jest ocena procesów (i ich dokumentacji), aktywności aktorów, detekcja anomalii, identyfikacji możliwości usprawnień oraz opracowanie metody analiz porównawczych projektów wykorzystujących różne technologie lub prowadzonych w ramach jednej organizacji. Część moich wyników było już opublikowane w pracach [10, 11, 14, 40].

Praca podzielona została na 3 główne sekcje. W rozdziale 3. przedstawiono metodykę analizy repozytoriów, dla których przeprowadzona została wielowymiarowa eksploracja cech takich jak: ogólnych charakterystyk, aktywności aktorów, profili komentarzy zgłoszeń oraz korelacji atrybutów zgłoszeń. Rozdział 4. pracy skupia się na wielowymiarowej eksploracji tekstów dostępnych w repozytoriach zgłoszeń. Wyszczególniłem w nim sekcje dotyczące analizy słownikowej opisów oraz opracowanych schematów klasyfikacji z uwzględnieniem niestandardowych obiektów tekstowych (obejmujących elementy wychodzące poza zakres słów języka naturalnego). W sekcji analizy słowników (4.2) przedstawiłem opracowane wyrażenia regularne identyfikujące te obiekty. Zostały one użyte w implementacji algorytmu przetwarzania tekstu podlegającego autorskim metodom klasyfikacji opisanym w sekcji 4.3. Dyskusję wyników eksperymentów klasyfikacji dwóch zadań klasyfikacji typów zgłoszeń i kategorii komentarzy zawarłem w sekcji 4.3.2.

Najbardziej obszerny rozdział 5. ukierunkowany jest na wielowymiarową analizę procesu obsługi zgłoszeń. W sekcji 5.1. przedstawiłem gruboziarnistą oceną efektywności procesu obsługi zgłoszeń poprzez wyznaczone profile czasowe obsługi zgłoszeń oraz liczby nierozwiązanych zgłoszeń. Zaprezentowałem również pojęcie długu błędów z algorytmem jego identyfikacji oraz sposobem minimalizacji ryzyka z nim związanego. Sekcja 5.2. zawiera opis autorskiego modelu IHG wraz ze szczegółowymi profilami oraz algorytmami. Sekcja 5.2.1. przedstawia szczegóły opracowanego modelu IHG, algorytmy analiz o niego oparte opisałem w sekcji 5.2.2. Sekcje 5.2.3. oraz 5.2.4. zawierają szczegółową charakterystykę profili stanów oraz ścieżek obsługi zgłoszeń opracowanych dzięki koncepcji IHG. Profile ścieżek zgłoszeń wykazały pewne anomalie procesu obsługi zgłoszeń, autorskie algorytmy wykrywania tych anomalii zawarłem w sekcji 5.2.5. Podsumowanie pracy wraz z perspektywą możliwości dalszych badań zawiera rozdział 6. Szereg dodatkowych i szczegółowych wyników badań zawarto w załącznikach 8.1, 8.2 oraz załączniku elektronicznym 8.3., 8.4.

3. Analiza repozytoriów

Wytwarzanie oprogramowania oraz jego utrzymanie wymaga śledzenia zgłoszeń w repozytoriach programowych. W szczególności istotne są raporty generowane w systemach ITS (ang. Issue Tracking Systems, np. Bugzilla, JIRA, Mantis) i kontroli wersji (np. Github, Bitbucket). Tematyka ta jest szeroko omawiana w literaturze [28, 38, 41, 42]. Systemy śledzenia zgłoszeń obejmują generowanie raportów o zarejestrowanych zgłoszeniach, stanie ich procesowania i postępie prac oraz decydującym rozwiązaniu (np. zakończone, naprawione, odrzucone, nie jest to problem, duplikat). Każde z zarejestrowanych zgłoszeń identyfikowane jest poprzez unikalny klucz, jednoznacznie opisujący raport w systemie. Wpis w repozytorium może być powiązany z poprawką, aktualizacją lub zmianą kodu (dodanie nowej funkcjonalności, scalenie, faktoryzacja kodu, zmiana konfiguracji itp.) zapisaną w systemie kontroli wersji (SVC, ang. *Software Version Control*). Repozytoria programowe są użyteczne w zarządzaniu rozwojem oprogramowania oraz związanymi aktywnościami utrzymaniowymi. Jednak z uwagi na presję czasu, dostępne informacje nie są zbadane w wystarczający sposób przez zaangażowanych aktorów oraz dostępne narzędzia. Z drugiej strony, badane są one w wielu pracach naukowych, głównie w odniesieniu do trzech aspektów: przewidywania błędów [43, 44], alokacji przypisania zgłoszenia do odpowiednich osób (ang. *triaging*) [45-47], a także wykrywania duplikatów zgłoszeń [45, 48]. Wiele badań koncentruje się na eksploracji opisów tekstowych problemów, w celu identyfikacji ich zasadności (błąd lub niewłaściwe zgłoszenie) lub określenia typu błędu [43, 49, 50]. Opracowane schematy analiz adresują wybrane unikalne problemy. Zakres badań ograniczany jest do konkretnych pytań, a analizy wyników pozbawione są szerszego kontekstu. Co więcej, autorzy bazowali na ogólnych (gruboziarnistych) cechach danych repozytoriów (płaskie modele, ang. *flat models*).

Analizując szeroki zakres repozytoriów projektów (*open source* i komercyjnych), zauważyłem, że dostępne dane wykorzystywane są w badaniach i praktyce zaledwie częściowo. Widoczna jest zatem potrzeba całościowych i obszernych eksploracji powiązanych ze sobą procesów wytwarzania i utrzymania oprogramowania. W tym celu opracowałem modele danych i schematy eksploracji, wsparte metrykami ukierunkowanymi na trzy cele badawcze:

- Ekstrakcja cech charakterystycznych (składniowych, semantycznych, czasowych i statystycznych) repozytoriów śledzenia problemów, odniesionych do różnych perspektyw obserwacji,

- Śledzenie efektywności obsługi zgłoszonych problemów, uwzględniając szeroki zakres zależności względem opracowanego modelu stanowego grafu (IHG),
- Wykrywanie niedoskonałości repozytoriów programowych oraz procesu obsługi zgłoszeń.

W badaniach zaadaptowałem i rozszerzyłem szereg statystyk i algorytmów eksploracji tekstu, uwzględniając specyfikę repozytoriów programowych, co znacząco odróżnia je od klasycznych metod eksploracji danych. Badając efektywność procesu obsługi zgłoszeń, używałem autorskich modeli grafów (IHG), które ułatwiają śledzenie przepływów obsługi zgłoszeń, aktywności aktorów, schematów obsługi różnych kategorii zgłoszeń etc. Przeprowadzone oceny dotyczą różnych perspektyw (globalne, lokalne), ujawniają korelacje danych z repozytoriów i zapewniają możliwość porównywania w ramach analizowanego projektu, kilku projektów jednej firmy lub wielu projektów z różnych środowisk programistycznych. Ekstrakcja danych oraz analizy wspierane są przez zaimplementowane narzędzie *IssueAnalyzerTool* oraz skrypty pomocnicze, których algorytmy opisane zostały w powiązanych sekcjach zaprezentowanych analiz.

3.1. Analiza literatury

Kwestie związane z rozwojem, utrzymaniem i jakością oprogramowania można ocenić, śledząc różne repozytoria, takie jak repozytorium zgłoszeń (ITS), repozytorium kodu (SVN), repozytoria testowe lub rekomendacje. Zapewniają one możliwość interakcji między uczestnikami projektu (programiści, użytkownicy), w celu tworzenia bądź komentowania błędów, zgłoszeń nowych funkcjonalności lub innych typów zadań. W literaturze dominują publikacje związane z niezawodnością i obsługą błędów. Na podstawie odstępu czasu między zgłoszonymi defektami lub liczby wykrytych problemów w kolejnych okresach możemy przewidzieć nieujawnione defekty. W tym celu opracowano i przeanalizowano wiele modeli wzrostu niezawodności, m.in. w [51-53]. Inne podejścia uwzględniają cechy oprogramowania (korelacja metryk złożoności kodu i gęstości defektów), procesu i problemu, np. [54]. Dostępne są także prace dyskutujące sposoby dobierania repozytoriów do analiz naukowych [55-57]. W [58] przedstawione zostało narzędzie wspierające eksplorację repozytoriów zgłoszeń, które dystrybuuje zadania do zdalnych agentów. Autorzy dokonali analizy możliwych usprawnień procesu eksploracji danych (*data minigu)* z wykorzystaniem opracowanego narzędzia.

Analizując raporty błędów, możemy prześledzić proces rozwoju projektu, role użytkowników [59], cykl życia oprogramowania [60], przepływ zgłoszeń [61] lub istotne

statystyki czasowe [62], istotności zgłoszeń[63]. Narzędzie do śledzenia zgłoszeń opisane w [62] pozwala na powiązanie zgłoszenia ze stosownym *commitem*, gałęzią kodu i ścieżkami plików. Procesy obsługi błędów analizowane są w publikacjach skupiających się na wybranych aspektach, takich jak: 1) wydajność procesu wspierana przez narzędzia do estymacji kosztów (zasoby ludzkie, czas) oraz określanie ryzyka poprzez stosowanie modeli symulacji [14], 2) detekcja duplikatów defektów (w celu usunięcia nadmiernych aktywności) [48], 3) alokacja zgłoszeń do rozwiązania do odpowiednich programistów (*bug triaging*). W [64] ryzyko wytwarzania oprogramowania oceniane jest przy użyciu modeli symulacji (ukierunkowane na schematy zwinne), bazujących na danych z systemu JIRA oraz cech pochodnych, takich jak aktywności rozwojowe bądź liczba zgłaszających. Z reguły, przypisywanie zgłoszeń do dalszego przetwarzania odbywa się poprzez stworzenie listy rekomendacji lub ranking programistów [65], organizację zespołów projektowych [66] oraz mapowanie programistów z raportami błędów [47].

Zgłoszenia mogą wymagać różnego procesowania, zależnie od swojego typu, określanego wyraźnie w odpowiednim polu repozytorium lub etykiecie. Etykietowanie może być używane do zapewnienia specyficznych informacji, rekomendacji dla danego zgłoszenia [47], wspomagania dalszego procesowania lub przyspieszenia rozwiązania. Często specyfikacja zgłoszenia bywa niepoprawna lub ominięta przez zgłaszającego. Można to potwierdzić, badając załączony opis zgłoszenia. W tym celu opracowane zostały dedykowane techniki eksploracji tekstu, ukierunkowane na wykrywanie kategorii ocenianych zgłoszeń. Odnoszą się one do rozróżniania błędów i innych zgłoszeń [50], duplikatów [45, 48], zgłoszeń problemów związanych z bezpieczeństwem [67, 68], niewymagających poprawki (ang. *wan't fix*) (np. z uwagi na przesunięcie do kolejnej wersji bądź znikomego wpływu) [69] oraz błędów związanych z pogorszeniem wydajności lub starzeniem się oprogramowania [70]. W [71] dyskutowane są klasy błędów np. *requirements, designe specification, user interface* itp. w referencji do czasu ich ujawnienia.

Sieci neuronowe RoBERT wykorzystywane są w [49] do klasyfikacji zgłoszeń w ramach trzech kategorii: błąd, usprawnienie, pytanie. W rzeczywistości nie jest to zgodne z zakresem kategorii zgłoszeń w różnych projektach. Jest to przeciwieństwo do podejścia opartego o słowa kluczowe. W [72] autorzy zajmują się problemem odrzucania lub zatwierdzania zgłoszeń do dalszego procesowania. Wykorzystują przy tym techniki przetwarzanie języka naturalnego do analizy sentymentu (pozytywny, negatywny) na podstawie często używanych fraz w podsumowaniu zgłoszenia. Każde przeprocesowane

zgłoszenie wraz z sentymentem konwertowane jest do cechy wektorowej wykorzystanej do klasyfikacji.

Zgłoszenia wzbogacane są o komentarze tekstowe, które zawierają pytania lub wyjaśnienia. Relacje między komentarzami oraz ich funkcje komunikacyjne, np. zdania referencyjne, rozkazujące, pytające, ekspresyjne lub emocjonalne, omawiane są w pracy [73]. Komentarze gromadzone są w wątkach dyskusji, co jest przydatne w analizach aktywności aktorów projektu. W pracy [74] zastosowano nadzorowaną technikę klasyfikacji, aby rozróżnić 16 rodzajów typów komentarzy. Trudno je uznać za uniwersalne w kontekście konkretnych projektów.

Niektóre prace proponują szereg schematów predykcji. Model zaprezentowany w [75] przewiduje pytania zadawane przez programistów. W [76] omówiono modele predykcji modułów oprogramowania potrzebnych do rozwiązania zadania, opracowane na podstawie danych historycznych raportów zgłoszeń. Autorzy [77] przedstawili narzędzie generujące model predykcji błędów na podstawie wskazanego repozytorium zgłoszeń, np. GitHub. W [78] przedstawiona została predykcja czasu rozwiązania zgłoszenia oraz analiza komponentów generujących najwięcej zgłoszeń błędów. Lokalizacja błędu w oparciu o podobieństwa opisów zgłoszeń błędów i kodów źródłowych omawiana jest w [79]. Inny model zaproponowano w pracy [43] do przewidywania celu zgłoszenia oraz jego priorytetu. Przewidywania krytyczności błędu omawiane są w [80], gdzie potwierdzono użyteczność połączenia cech z nieustrukturyzowanego tekstu i metod eksploracji tekstu. W praktyce występują różne niedoskonałości w raportowaniu zgłoszeń, wynikające głównie z presji czasu, przeciążenia interesariuszy projektu, ich zaniedbań itp. Niedoskonałości (zwane *bad smells*) zgłoszeń błędów wymienione zostały w [81], bazując na przeglądzie literatury oraz wynikach ankiety z kilku firm. Były to między innymi: brak linku do poprawki w kodzie, niepełne rozwiązanie, brak przypisania błędu do osoby obsługującej, niepoprawny poziom ważności, brak informacji o środowisku.

Wymienione prace potwierdzają mnogość aspektów rozwoju projektów, które można wyróżnić analizując repozytoria oprogramowania. Zaproponowane metody zostały jednak ograniczone do wybranych danych z repozytoriów i ukierunkowane są na analizę konkretnych problemów z pominięciem pozostałych aspektów. Badania nad napływem i przetwarzaniem zgłoszeń mają charakter powierzchowny i ograniczają się do podstawowych statystyk. Analizując szeroki zakres repozytoriów programowych, zidentyfikowałem wiele szczegółów zgłoszeń, które były pomijane w badaniach, a które zapewniają dokładniejszy wgląd

w schematy procesu obsługi zgłoszeń. Dokonałem również ich wstępnej ewaluacji poprzez metody *text minigu* w publikacji [40], zaprezentowanej na konferencji ENASE 2023. Potwierdza to potrzebę holistycznej i drobnoziarnistej eksploracji repozytoriów programowych w celu poprawienia systematycznej oceny projektów w różnych perspektywach.

3.2. Charakterystyka repozytoriów programowych

Systemy śledzenia zgłoszeń (ITS) wspierają cały cykl życia projektu. Najczęściej wykorzystywane są do tworzenia, aktualizacji oraz rozwiązywania zgłoszeń przez aktorów projektów. Interesariusze projektu, często określani jako aktorzy, zaangażowani są w rozwój, testowanie oraz użytkowanie końcowego oprogramowania. Posiadając dostęp do systemu ITS, mogą oni uzupełniać raporty zgłoszeń zgodnie z atrybutami i dostępami nadanymi przez administratorów systemu, którzy często są również managerami projektów. Dostępnych jest wiele rozwiązań komercyjnych oraz typu *open source* z klasy ITS, które różnią się szczegółami dotyczącymi raportowania zgłoszeń (np. typy zgłoszeń, opcje rozwiązania, poziomy istotności, stany obsługi) oraz funkcjonalności (głównie w postaci użyteczności systemu). Systemy klasy ITS posiadają również podklasę systemów określanych jako systemy śledzenia błędów/defektów (BTS, ang. *Bug Tracking Systems*), w których śledzone są wyłącznie błędy zgłaszane w trakcie projektu. Przykładem takiego systemu jest Bugzilla bądź Mantis Bug Tracker.

Zazwyczaj narzędzia klasy ITS pozwalają na elastyczność w dostosowywaniu pól obowiązkowych, rekomendowanych bądź opcjonalnych do potrzeb projektu w organizacji. Możliwe jest także zdefiniowanie dedykowanych pól o określonych typach, jak i dedykowanych typów zgłoszeń. Skutkuje to różnorodnością w zawartościach repozytoriów. Niemniej, możliwe są do zastosowania pewne formy abstrakcji lub uogólnienia mapowania danych w celu umożliwienia porównań między projektowych. Ponadto, w projektach wieloletnich zaobserwowałem zmiany w procesie raportowania zgłoszeń, np. zmiana używanych typów zgłoszeń, kategorii, stanów, co utrudnia proces analizy. Obserwacje te przedstawione są w szczegółach w dalszej części pracy na podstawie wybranych projektów raportowanych w najbardziej popularnym narzędziu klasy ITS – JIRA.

Zgłoszenia dodane do repozytorium powinny być przetworzone do stanu końcowego z określonym sposobem rozwiązania, np. zakończone, poprawione, odrzucone jako nie problem, duplikat. Sam proces obsługi zgłoszeń może składać się z kilku faz identyfikowanych przez stany, np. nowe zgłoszenie, analiza, w trakcie przetwarzania, rozwiązane, sprawdzone, zamknięte. Proces ten udokumentowany jest w repozytorium, jednak dostęp do

tak szczegółowych danych nie jest trywialny i zależy od typu zgłoszenia, lokalizacji i innych specyfikacji. W uogólnieniu możemy wyróżnić 6 kategorii pól w repozytoriach, bazując na systemie JIRA:

- *Identyfikacja zgłoszenia*: nazwa projektu, unikalne ID zgłoszenia, etykieta
- *Definicja zgłoszenia*: tytuł zgłoszenia, opis zgłoszenia, podsumowanie zgłoszenia, typ zgłoszenia, priorytet, istotność
- *Postęp w procesowaniu zgłoszenia*: status, rozwiązania, dodane komentarze
- *Aktorzy uczestniczący w zgłoszeniu*: reporter (osoba tworząca zgłoszenie), osoba aktualnie przypisana do obsługi zgłoszenia (ang. *Assignee*), osoby obserwujące zgłoszenie (ang. *Watchers*)
- *Lokalizacja zgłoszenia:* wpływ na wersję (ang. *Affected Versions*), wersja rozwiązania (ang. *Fix Version*), komponenty (ang. *Components*), środowisko (ang. *Enviroment*)
- *Podstawowe parametry czasowe*: data dodania zgłoszenia (ang. *Creation Date*), data rozwiązania zgłoszenia (ang. *Resolution Time*), data ważności zgłoszenia (ang. *Due Date*)
- *Dodatkowe informacje*: szacowany czas niezbędny do rozwiązania zgłoszenia (ang. *Estimates*), załączniki

W repozytoriach projektów komercyjnych mogą występować dodatkowe pola, np. w projekcie badanym w Instytucie Informatyki ZOiAK [82] liczba używanych pól wynosiła ponad 100. W szczególności zawierały one referencje do testów (scenariusze testowe, kategorie, grupy, ścieżki), które w innym projekcie [83] przechowywane były w dedykowanym repozytorium. Z drugiej strony, wiele pól było rzadko używanych lub zawierały tylko jedną wartość. Zakres danych zawartych w repozytoriach ITS jest potencjalnie duży i zasadne wydaje się sprawdzenie ich użyteczność w całościowej i głębszej ocenie projektów. Praca skupia się na pozyskaniu lepszej widoczności tych danych oraz eksploracji ich wartości, jak również odkryciu różnic w nich występujących.

Poszczególne atrybuty zgłoszeń uzupełniane są przez aktorów z różną precyzją. Wpływ na to ma obowiązkowość pola, poziom skomplikowania związany z uzupełnieniem informacji lub restrykcje walidacji informacji. Istotnym czynnikiem jest również stałość zespołu, rozumiana poprzez poziom rotacji poszczególnych aktorów w projekcie. Co więcej, poszczególni aktorzy mogą wykazywać różny poziom motywacji do uzupełniania informacji oraz aktualizacji ich aktywności w odpowiedni sposób w repozytoriach programowych.

Możliwe jest także występowanie żargonu, skrótów bądź nazw projektowych w polach tekstowych. Dodatkowo pola te obarczone są błędami w pisowni, gramatyce lub zawierają przeploty różnych języków. Niedoskonałości danych w repozytoriach oraz ich niespójność wymagają większego wysiłku przy ich eksploracji i wyciąganiu wniosków. Problematyczna jest także ocena niepoprawnych danych, ich duplikacji i spójności raportów.

Ważnym zagadnieniem jest określenie cech wartościowych w ocenie projektu. Niektóre cechy mogą być wyznaczone poprzez proste pobranie wartości pola zgłoszenia, inne zaś wymagają głębszej analizy dostępnych danych, np. połączenia kilku pól przy odpowiednich warunkach lub wyznaczenia określonych cech na podstawie pobranych danych. Występują również przypadki kiedy niezbędne jest pobranie informacji nie tylko z poszczególnych pól, ale z różnych repozytoriów, np. z repozytorium kodu projektu. Badania przedstawione w pracy wsparte były przez szereg skryptów oraz program implementujący wszystkie algorytmy niezbędne do: 1) wyznaczenia danych statystycznych i jakościowych zawartości repozytoriów, 2) eksploracji danych pól tekstowych uwzględniając szereg kategorii słowników oraz korelacji, 3) śledzenia procesu obsługi zgłoszeń. Niektóre systemy ITS pozwalają na wykorzystanie serwisów API w celu tworzenia nowych zgłoszenia, aktualizacji ich, ale także pobierania danych już istniejących zgłoszeń. Dostęp do danych z systemu jest niezbędny w celu przeprowadzenia szczegółowych analiz. Implementacja generycznego rozwiązania, pozwalającego na wsparcie prezentowanych analiz, wymagała zastosowania pewnych uogólnień oraz rozwiązania problemów w dostępie do danych.

Poważnym utrudnieniem pobierania informacji o zgłoszeniach z systemów ITS poprzez serwisy API jest wydajność. Czas odpowiedzi serwisu jest zadowalająca w przypadku pobierania informacji o jednym zgłoszeniu, jednak rośnie on liniowo wraz ze zwiększeniem zakresu zapytania. Przykładowo, czas pobrania 5 tys. zgłoszeń może wynieść ponad 30 min. Wynika to z zabezpieczeń API JIRA, które domyślnie ograniczają liczbę zwracanych zgłoszeń poprzez API do 100. W celu pobrania większej liczby zgłoszeń niezbędne jest zastosowanie paginacji, tzn. podzielenie wyszukania na X części oraz dodanie do wywołania serwisu zakresu indeksów początku i końca szukanego wyniku. Dane wykorzystywane w pracy do wielowymiarowych analiz rozdzielone są w JIRA na różne serwisy API. Szczegóły zgłoszenia dostępne są w jednym serwisie, a komentarze zwracane są innym. Taki podział struktury danych na których bazują serwisy API zaobserwowałem zarówno w JIRA, jak i narzędziu Bugzilla. Oznacza to, że w celu zbudowania pełnego modelu danych dla wszystkich pobranych zgłoszeń niezbędne jest wykonanie pojedynczych wywołań serwisu API zwracającego

komentarze dla każdego pobranego zgłoszenia. Dla ilustracji, kolejność oraz liczby wywołań serwisów JIRA API dla 1000 algorytm pobieracjący pełen model danych możemy zapisać następującymi krokami: 1) Wykonaj 10 wywołań serwisu search wraz z paginacją (100 wyników na jednej stronie), 2) Dla każdego zgłoszenia z pobranej listy zgłoszeń wykonaj zapytanie serwisu API komentarzy. Sumaryczna liczba wywołań dla 1000 zgłoszeń wynosi 1010.

Innemy problemem przy pobieraniu danych z różnych instalacji narzędzia JIRA jest format dat oraz typ sortowania względem niej, np. komentarzy czy historii zmian statusów. Format daty jest cechą konfigurowalną. Dla przeprowadzania wartościowych analiz czasowych dokonuję standaryzacji pobieranych dat do jednego wspólnego formatu. Standaryzacji zostały poddane również listy zwracane w odpowiedzi serwisów, które sortowane są względem dat, np. od najstarszych do najnowszych bądź od najnowszych do najstarszych. Typ sortowania określany jest przez administratora systemu danej instancji JIRA. Niepoprawne sortowanie powodowało błędy w wyliczaniu czasu przejścia między stanami bądź czasu dodania poszczególnych komentarzy. Implementacja oparta o serwisy API JIRA związana jest z ryzykiem zmiany ich działania

Wykorzystanie zewnętrznych serwisów API niesie ze sobą również ryzyko związane z możliwymi zmianami po stronie systemu źródłowego (JIRA) oraz konfiguracji repozytoriów ITS. Repozytoria ITS projektów Open Source są otwarte dla użytkowników, jednak podlegają ciągłym modyfikacjom, np. możliwe jest ograniczenie dostępności serwisów API dla anonimowych użytkowników. W takim przypadku niezbędne byłoby stworzenie konta użytkownika oraz uwierzytelnienie wywołań serwisów API. Aplikacja JIRA jest również w ciągłym rozwoju, co znacząco wpływa na używane standardy API, które mogą stopniowo wychodzić z użytku (ang. *deprecated*). Przykładowo, podczas prac badawczych w systemie JIRA nastąpiła zmiana związana ze sposobem autoryzacji serwisów API. W początkowej fazie badań, autoryzacja opierała się o zaszyfrowane parametry *userID* i *password*. Po aktualizacji wersji JIRA uwierzytelnienie tym sposobem było już niemożliwe. Nowa wersja systemu wymusiła użycie protokołu Oauth 2.0, które bazuje na nazwie użytkownika i kodzie API generowanym w panelu użytkownika JIRA.

Rozważania prowadzone w pracy oparte są na danych z szeregu projektów *open source* i komercyjnym raportowanych w narzędziu JIRA. Pod uwagę brane były zgłoszenia nowych funkcjonalności (włączając zadania techniczne - *TASK*) i błędów. Dla przejrzystości pracy

główne analizy przedstawione są na wybranych projektach. Całość zagregowanych danych dostępna jest w elektronicznym załączniku pracy (opis w sekcji 8.3).

3.3. Wielowymiarowa eksploracja cech repozytoriów

Wstępna ocena repozytoriów ITS wykazała różnorodność zawartych w nich danych, które wykorzystywane są w ograniczony sposób w codziennej pracy oraz w prowadzonych badaniach. Większość z nich jest pomijana bez sprawdzania ich użyteczności w całościowej ocenie projektów. Stąd wynika potrzeba przeprowadzenia dokładniejszych i bardziej usystematyzowanych badań repozytoriów, uwzględniając szereg różnorodnych perspektyw wraz z propozycją pewnych kategorii klasyfikacji. W pracy skupiłem się na 3 typach eksploracji:

1. Ogólne raporty statystyczne wraz z czasami aktywności
2. Strukturalne i semantyczne analizy podstawowych cech raportowanych pól
3. Określenie różnic repozytoriów oraz ich niedoskonałości

W ramach badań pkt.1 biorę pod uwagę ogólne cechy repozytorium, takie jak przyrost liczby zgłoszeń w poszczególnych okresach czasu (możliwa jest do zastosowania perspektywa dni, miesięcy, lat), trendy krótko- i długoterminowe, przyrost konkretnych danych (np. zgłoszenia specjalnego typu bądź komentarze), anomalie, przyrost nierozwiązanych zgłoszeń. Cechy te mogą być rozszerzone poprzez zastosowanie pkt.2 z listy, gdzie skupiam się na strukturze repozytorium i ważności poszczególnych informacji, w szczególności wypełnienia raportowanych pól, możliwych wartości w przypadku list lub słowników oraz ich ważności. Cechy te mogą być zależne od typu zgłoszenia oraz konkretnej konfiguracji repozytorium projektowego lub polityki organizacji. Analizy pokazują użyteczność poszczególnych pól oraz ich zmiany w czasie. Możliwe jest przeprowadzenie dedykowanej agregacji pól w czasie oraz zaobserwowania trendów, anomalii bądź dystrybucji poszczególnych wartości w różnych okresach czasowych. Cechy te mogą być wyznaczone na podstawie poszczególnych pól raportów zgłoszeń.

Bardziej szczegółowe badania zaprezentowane w dalszej części pracy skupiają się na odnalezieniu korelacji między danymi oraz ich zależności. W szczególności interesujące są profile zaangażowania aktorów w projekcie, które możemy ocenić między innymi z perspektywy czasowej oraz aktywności w poszczególnych aktywnościach, jak np. tworzenie zgłoszeń bądź ich komentowanie, fluktuację aktorów w czasie trwania projektu. Pozyskane dane możliwe są do prezentacji w formie statystycznej (tabele) lub graficznej (wykresy i grafy).

W ramach pracy zaproponowałem pewne sposoby ich agregacji, co przedstawione jest w dalszej części pracy.

Pewne dyskusje w tekście skupiają się na określeniu różnic pomiędzy repozytoriami w poszczególnych projektach bądź wewnątrz organizacji. Kwestia ta może być istotna dla przeprowadzenia oceny porównawczej projektów prowadzonych w tej samej firmie.

3.3.1. Charakterystyki ogólne

Ogólne statystyki wartości pól repozytorium możemy przedstawić za pomocą następujących cech: współczynnik wypełnienia poszczególnych pól (w procentach), rozkład przyjmowanych wartości, najrzadziej i najczęściej występująca wartość w polu, liczba możliwych wartości, cechy czasu dla zgłoszeń (kwartyle Q1, Q2, Q3). Wartości te będą zależne od kategorii poszczególnych pól, których ogólny podział przedstawiony został na początku rozdziału. Dla przykładu, powyższe charakterystyki oraz ich dystrybucja nie są interesujące dla pól identyfikujących zgłoszenie, ponieważ zawierają one unikalne wartości. Wartościowe są statystyki pól z dużą liczbą wartości, takie jak np. pole *Fix Version, Affected Version's, Components, Enviroment* (lokalizacyjne). Pola te pozwalają na lepszą lokalizację zgłoszenia, w szczególności w przypadku zgłoszeń błędów, gdzie jasno wskazują wersję, w której występuje błąd (*Affected Version*) lub środowisko (*Enviroment*). Ustawiane są one najczęściej przez reportera lub w trakcie procesowania zgłoszenia przez osobę obsługującą zgłoszenie. Pole *Fix Version's* powinno być natomiast uzupełnione przez osobę rozwiązującą zgłoszenie, ponieważ zawiera informację o wersji, w której dostarczone będzie rozwiązanie (poprawka) zgłoszenia. Widoczne jest zatem, że pola te są szczególnie istotne dla zgłoszeń błędów, a niekoniecznie dla zgłoszeń nowych funkcjonalności. Współczynnik ich wypełnienia może poprawić diagnostykę.

Pola tekstowe zawierają różne opisy, które mogą znacząco się różnić w zależności od typu zgłoszenia. W tym przypadku można badać rozkłady objętości opisów, ich cechy syntaktyczne i semantyczne. Wartościowe jest użycie metryk opartych na eksploracji tekstu, takich jak rozkład wykorzystanych słów (język naturalny, słowa techniczne, symbole itp.). Opracowane statystyki mogą być zastosowane dla zbioru zgłoszeń, niezależnie od typu lub dotyczyć wyszczególnionych typów, priorytetów, profili (programista, tester) tworzących zgłoszenie, co daje o wiele lepsze zrozumienie danych zawartych w repozytorium oraz procesu obsługi zgłoszeń.

Dla większości zweryfikowanych repozytoriów projektów liczba pól w zgłoszeniach nie przekracza 25, jednak zazwyczaj tylko dla około 10 pól współczynnik wypełnienia

jest większy niż 90%. W większości wypadków zgłoszenia posiadają uzupełnione pola obowiązkowe bądź rekomendowane.

Pierwszy podstawowy parametr charakteryzujący repozytorium składa się z dwóch współczynników: współczynnik wypełnienia i liczby możliwych wartości w polach kategorycznych. Przykład takich współczynników dla wybranych projektów przedstawiłem w Tab. 1 oraz Tab. 2. Tab. 2 opracowana została na podstawie zagregowanych danych z [84] dla projektów Apache Casandra (cas), Apache Spark (spark), Apache Flink (flink), Apache Ignite (ignite), Mozilla Tunderbird (moz), Red Hat Enterprise Linux 8 (red). Liczba weryfikowanych zgłoszeń podana jest w nawiasach pod nazwą projektów w Tab. 1 oraz Tab. 2. W prezentacji pominięte zostały pola obowiązkowe z wypełnieniem 100% oraz pola z współczynnikiem 99% - 100%, takie jak identyfikator, priorytet, typ zgłoszenia, status, tytuł, reporter, data stworzenia zgłoszenia itp. Pola te są obowiązkowe, ale mimo to, w niektórych repozytoriach występują niespójności danych, jak np. brak wartości w obowiązkowym polu. Źródło tego problemu jest trudne do zdefiniowania, jednak dotyka bardzo małej liczby zgłoszeń. Innym przykładem może być pole Opis zgłoszenia (współczynnik wypełnienia: 89.8 – 99%), które dla niektórych zgłoszeń bywa puste. Zazwyczaj jednak w takich przypadkach tytuł w połączeniu z podsumowaniem zgłoszenia jednoznacznie określa zgłoszenie. Istotną obserwacją jest mały procent wypełniania pól w projekcie P1 (projekt komercyjny z moim istotnym udziałem), który wskazuje na pewne braki w raportowaniu. Sytuacja ta związana jest z zastosowanym podejściem projektowym, w którym wszystkie zadania są priorytetyzowane i rozwiązywane względem priorytetu, a wskazanie dostarczenia określane jest w polu *Label*. Jest to cecha charakterystyczna dla projektu P1. W projektach Log4J2, Spark, thun, red widoczna jest natomiast wysoka precyzja określenia powiązania z komponentem odpowiednio 98,5%, 95,5%, 100%, 100% co może być związane ze specyfiką techniczną projektu. Osoby raportujące zgłoszenia błędów w projektach *open soruce* są zazwyczaj wykwalifikowane technicznie, co pozwala im na określenie powiązanego ze zgłoszeniem komponentu. Atrybut *Enviroment* jest polem tekstowym, w projektach MongoDB, natomiast w projekcie P1 został on wyłączony poprzez konfigurację narzędzia.

Atrybut	Groovy (6514)	Lucene (8685)	Log4J2 (3108)	Arrow (15773)	MongoDB (13434)	P1 (2378)
Resolution	85,5%/22	80,96%/22	75,5%/22	84,97%/22	87,9%/10	40,16%/14
Fix Vers.	66,96%/258	66,46%/183	59,6%/74	72,9%/53	66,59%/634	38,7%/377
Aff Vers.	73,1%/258	37,2%/183	75,45%/74	21,1%/53	12,9%/634	38,1%/377
Compon.	62,8%/46	56,35%/46	79,9%/39	98,5%/34	34,8%/47	-
Environ.	24%	7,75%	26,1%	5,5%	-	-

Tab. 1 Współczynnik procentowego wypełnienia wybranych atrybutów zgłoszenia i liczba przyjmowanych wartości

Atrybut	Flink (28714)	Ignite (17396)	Spark (39500)	thun (62207)	red (22066)
Resolution	83,0%/19	74,2%/19	92,7%/19	86,6%/9	88,4%/13
Fix Vers.	62,1%/195	61,1%/68	59,5%/324	-	43,3%/6605
Aff Vers.	-	42,3%/168	76,3%/1176	100%/108	100%/11
Compon.	50,5%/450	63,0%/261	95,5%/620	100%/26	100%/1387
Environ.	15,4%/2252	2,2%/355	7,1%/324	-	0,05%/12

Tab. 2 Współczynnik procentowego wypełnienia wybranych atrybutów zgłoszenia i liczba przyjmowanych wartości (opracowane na podstawie [84])

Ważną cechą repozytoriów jest rozkład wartości (kategorii) w odpowiednich polach. Przykład takiej analizy przedstawia Tab. 3. Zawiera ona rozkład najważniejszych kategorii w polach: priorytet, rozwiązanie, typ zadania, status procesu. Dla lepszej prezentacji zastosowałem dwie kategorie: *Closed* (zgłoszenie dotarło to stanu finalnego) i *Open* (inny status niż status terminalny w projekcie). Standaryzacja została także wprowadzona w polu Priorytet. Wartości oznaczające priorytet są także modyfikowalne i mogą różnić się między projektami. Projekt MongoDB używa adnotacji priorytetów „*Blocker – P1”*, „*Critical – P2”*, „*Major – P3”*, „*Minor – P4”*, „*Trivial – P5”*. Projekt Log4J2 wykorzystuje jedynie nazwy priorytetu jako „*Blocker”*, „*Critical”*, „*Major”*, „*Minor”*, „*Trivial”*.; projekt komercyjny natomiast: „*Urgent”*, „*High”*, „*Medium”*, „*Low”*, „*Cosmetic”*, gdzie „*Blocker – P1”*, *„Blocker”*, *„Urgent”* oznaczają najwyższy priorytet. Dla ujednolicenia analiz priorytety zostały

ujednolicone do wartości *Blocker, Critical, Major, Minor, Trivial*. Dodatkowo z uwagi na niskie użycie priorytetów *Minor i Trivial* występują one w jednym wierszu, reprezentując sumę pokrycia tych stanów. Wartości pominięte w Tab. 3 stanowią 100 – S%, gdzie S% to suma poszczególnych wartości % w komórce. Typ *User Story* dla ułatwienia prezentacji został zmapowany do wartości *Improvement*, ponieważ nie występuje on w projektach *open source*.

Projekt (liczba zgłoszeń)		Groovy (6514)	Lucene (8685)	Log4J2 (3108)	Arrow (15773)	MongoDB (13434)	P1 (2378)
Priorytet	Blocker	3,12	4,05	5,65	6,06	0,68	1,72
	Critical	5,75	2,19	6,71	4,86	1,53	14,21
	Major	73,35	64,84	72,79	73,06	94,86	33,60
	Minor, Trivial	17,80	28,92	14,84	16,01	2,93	50,46
Typ	Bug	75,51	45,86	61,00	38,34	62,57	65,39
	New Feature	4,96	46,69	9,75	12,30	5,75	9,84
	Improvment	19,54	7,45	29,25	49,36	31,65	10,39
Status	Open	14,60	19,15	24,58	15,03	13,10	13,41
	Closed	85,40	80,85	75,42	84,97	86,90	86,59

Tab. 3 Współczynnik procentowy wypełnienia wartości kategorii atrybutów Priorytet, Typ, Status.

Rozkład wartości zaprezentowanych pól może różnić się znacząco między projektami w zależności od konfiguracji narzędzia (dostępne wartości pól, wartości domyślne), polityki organizacji, aktywności aktorów. Najwięcej zgłoszeń w projekcie MongoBD przypisanych było do priorytetu *Major* (94%). Jednak dominacja tego statusu jest widoczna również w innych projektach *open source*. Projekt komercyjny P1 jako jedyny wykazuje malejącą liczbę zgłoszeń wraz z rosnącym priorytetem zgłoszenia, co świadczy o dobrej jakości projektu i małej fluktuacji zespołu, który dobrze zna dostarczane rozwiązanie i może zminimalizować liczbę zgłoszeń o krytycznym priorytecie. Dominacja priorytetu Major w projektach OS może wynikać z wartości domyślnej jaka przypisywana jest do zgłoszenia podczas jego tworzenia w badanych repozytoriach. Reporter może zmienić tę wartość, jednak jest ona najbardziej uniwersalna w subiektywnym odczuciu. Specyficzna dystrybucja wartości pola Priorytet w projekcie komercyjnym P1, może być uzasadniona wieloletnią współpracą zespołu, który cechuje się małą rotacją. Wpływa to znacząco na dobre zrozumienie reprezentacji poszczególnych priorytetów zgłoszeń. Analiza dystrybucji wartości pól kategorii może uwzględniać perspektywę czasową, dla której możemy zastosować czas poszczególnych

sprintów dla projektów *SCRUM*-owych oraz typ zgłoszenia. Dla przykładu, w projekcie komercyjnym P1 dystrybucja wartości dla pola Priorytet w okresie trzech następujących po sobie sprintów wynosiła: dla wszystkich zgłoszeń (584): 2.4%, 19.0%, 34,4%, 41,9%, 2.2%; a dla samych zgłoszeń błędów: 1.9%, 21.5%, 28.5%, 45%, 2,4%. Wartości te zostały przedstawione dla kolejnych priorytetów: *Blocker, Criticla, Major, Minor, Trivia*. Analizując rozkład wartości kategorii atrybutów zgłoszeń możemy uwzględnić również atrybut Resolution określający sposób rozwiązania zgłoszenia. Dla przykładu w projekcie komercyjnym P1 sposób rozwiązania może być określony na 13 sposobów, jednak pole nie jest obowiązkowe. Spośród przeanalizowanych zgłoszeń (projektu P1 z wybraną wartością w polu *Resolution*), 81,2% zgłoszeń miało ustawioną jedną z 4 wartości określających sposób rozwiązania jako pomyślany, 15,7% jedną z 8 wartości określających odrzucenie zgłoszenia, oraz 3,1% zgłoszeń miało przypisaną wartość kategorii Resolution określającą duplikat. Rozkład wartości kategorii pola Resolution jest lepszy niż wartości zaprezentowane w [84] dla projektów *open soruce* w których procent użycia wartości *Duplicate* przekracza 6,1% z maksymalną wartością 10,3 dla projektu Apache Casandra. Liczba możliwych sposobów rozwiązania (liczba możliwych wartości w polu Resolution) w projektach była zbliżona, w zakresie 7-14.

Dystrybucję typów zgłoszeń projektów z Tab. 3 możemy przedstawić w formie wektorów gdzie kolejne wartości liczbowe odpowiadają zgłoszeniom określonego typu. Dla zgłoszeń projektów *open source* uwzględniłem 3 typy zgłoszeń *New Feature, Improvement, Bug* których dystrybucja wygląda następująco: *Groovy* {323, 1273, 4919}, *Lucene* {646, 4055, 3983}, Log4J2 {303, 909, 1896}, Arrow {1940, 7786, 6047}, MongoDB {773, 4257 , 8405} gdzie wartości przedstawiają odpowiednio liczbę zgłoszeń typu: *New Feature, Improvement, Bug*. W projekcie komercyjnym P1 dystrybucja zgłoszeń odpowiada charakterystyce {234, 247, 342, 1555}, której kolejne wartości przedstawiają liczbę zgłoszeń dla typów *New Feature, User Story, Task, Bug.* Rozkład typów zgłoszeń może być pewnym wyznacznikiem fazy projektu oraz jakości prowadzonych testów. Projekt P1 znajduje się w fazie ciągłego rozwoju z nakładem prac ukierunkowanym na polepszenie jakość rozwiązania (większa liczba zgłoszeń błędów). Podobnie wygląda rozkład zgłoszeń projektu Groovy, oraz MongoDB.

Niektóre wartości pól raportów zgłoszeń nie powinny podlegać zmianie w czasie przetwarzania zgłoszenia, np. priorytet czy typ zgłoszenia. Wystąpienie takich anomalii powinno być wykryte i przeanalizowane. Lista możliwych wartości dla pól kategorii w badanych projektach wydaje się być stałą wartością, jednak w bogatym doświadczeniu zawodowym miałem okazję zaobserwować pewne anomalie. Mogą one wynikać ze zmiany

konfiguracji repozytorium, organizacji prac zespołu lub wersji systemu ITS bądź błędów systemu.

W prowadzonych rozważaniach stosuję zamiennie określenie atrybut w odniesieniu do określenia pole. W każdym z badanych projektów oprócz atrybutów domyślnych występują również pola dodatkowe (zależne od konfiguracji danego projektu). Podczas prowadzonych analiz zauważyłem że często są one zależne od typu zgłoszenia, niemniej współczynnik ich wypełnienia jest różny (w przypadku automatycznego uzupełniania stopień wypełnienia równy jest 100%).

Analiza ITS może być ukierunkowana również na kwestię optymalnej konfiguracji narzędzia, polegającej na ocenie użyteczności dodatkowych atrybutów zgłoszeń. Dla ilustracji wykonałem taką statystykę dla dwóch projektów MongoDB oraz projektu komercyjnego P1. W projekcie MongoDB dla zgłoszeń typu *New Feature, Improvement, Bug* zdefiniowano 52 dodatkowe atrybuty z których 13 wypełnionych jest w 100% (uzupełniane automatycznie) i 6 z wypełnieniem 0%. Wybrane pola konfigurowalne z ciekawą specyfiką przedstawiłem w Tab. 4, gdzie dla ułatwienia prezentacji nazwy pól (pierwszy wiersz tabeli) zostały skrócone do postaci np. *C12751* która równoznaczna jest nazwie technicznej pola *customfield_12751*. Ustalenie biznesowych nazw pól w projektach *open soruce* jest trudne z uwagi na małą znajomość specyfiki projektu oraz ograniczenia możliwości użycia serwisów API przez osoby nie będące członkami zespołu. Dlatego poniższe analizy oparte są o techniczne nazwy pól. Liczba rozpatrywanych zgłoszeń podana jest w nawiasach pod każdym z typów zgłoszenia.

Typ zgłoszenia (liczba zgłoszeń)	C12751	C18254	C10032	C17050	C16465	C12450	C17051	C10555	C23361
Łącznie (13434)	19,86%	6,85%	60,28%	67,05%	34,59%	25,71%	2,69%	8,72%	0,01%
New Feature (773)	32,08%	23,80%	-	57,83%	11,25%	12,42%	4,92%	19,53%	0,26%
Improvement (4257)	33,78%	9,58%	-	58,75%	16,54%	15,76%	4,02%	9,09%	0,02%
Bug (8405)	11,68%	3,91%	96,34%	72,10%	45,88%	31,98%	1,82%	7,55%	-

Tab. 4 Współczynnik wypełnienia wybranych atrybutów zgłoszeń projektu MongoDB

Na podstawie opracowanej agregacji widoczna jest różnica w postaci sposobu wypełniania pól dodatkowych w zależności od typu zgłoszenia, przykładem mogą być pola *C16465, C16465* które są najczęściej uzupełniane w zgłoszeniach błędów (ang. *Bug*). Występują również pola które nie są dostępne w określonych typach *C10032* (niedostępny w zgłoszeniach typu *Improvement*) lub *C23361* (brak dla typu *Bug*).

Projekty prowadzone zgodnie z metodologią *SCRUM* różnią się znacząco od projektów *open source*. Tab. 5 przedstawia współczynnik wypełnienia wybranych atrybutów zgłoszeń projektu komercyjnego P1 prowadzonego w *SCRUM*, gdzie techniczne nazwy atrybutów zgłoszenia przedstawione zostały w pierwszym wierszu, a liczba rozpatrywanych zgłoszeń w zależności od typu zgłoszenia podana jest w nawiasach pod nim (pierwsza kolumna).

Typ zgłoszenia (liczba zgłoszeń)	C10322	C10542	C12800	C12986	C12835	C12834	C12995	C12855	C12857
Łącznie (2378)	18,97%	18,46%	9,08%	9,84%	9,80%	9,80%	8,87%	0,08%	0,08%
New Feature (234)	-	-	3,42%	100,00%	99,57%	99,57%	44,44%	0,43%	0,43%
User Story (247)	84,62%	84,62%	83,00%	-	0%	0,40%	0,40%	0,40%	0,40%
Task (342)	70,47%	67,54%	0,88%	-	-	-	0,58%	-	-
Bug (1555)	0,06%	-	-	-	-	-	6,75%	0,06%	0,06%

Tab. 5 Współczynnik wypełnienia wybranych atrybutów zgłoszeń projektu komercyjnego P1

W projekcie P1 widoczna jest znacząca korelacja między stopniem wypełnienia atrybutu zgłoszenia a jego typem przykładem może być pole *C12800* które najwyższy stopień wypełnienia ma dla typu *User Story*. Zauważalna jest także duża różnica pomiędzy liczą dodatkowych pól w zależności od typu zgłoszenia, gdzie najwięcej pól dodatkowych skonfigurowanych zostało dla zgłoszeń typu *User Story*. W projekcie komercyjnym P1 występuje łącznie 39 pól dodatkowych (łącznie dla typów zgłoszeń *New Feature, User Story, Task, Bug*) z których dla 9 współczynnik wypełnienia wynosi 100%, a 6 pozostanie nie wypełnione w żadnym zgłoszeniu. Liczba pól dodatkowych w zależności od typu zgłoszenia wynosi odpowiednio 26 (*New Feature*), 34 (*User Story*), 20 (*Task*), 23 (*Bug*). Pola konfigurowalne w projekcie komercyjnym P1, odpowiadają za wzbogacenie zgłoszenia o informacjach dotyczących lokalizacji błędu (np. środowisko testowe) lub estymacji złożoności prac, nazwy biznesowe pól nie mogły zostać udostępnione w pracy dlatego zastąpione przykładowymi nazwami.

Współczynnik wartości atrybutu może być także skorelowany z typem danych w nim zawartych, poprzez profil wartości atrybutu $At|AtU$ opisanego przez wzór:

$$At|AtU = \{\text{typ atrybutu; procent wypełnienia atrybutu; } \underline{\text{liczba możliwych wartości}}\text{; <rozkład procentowy przyjmowanych wartości>}\} \quad (3.1)$$

gdzie *At* oznacza nazwę atrybutu zgłoszenia, a *AtU* określa typ danych atrybutu z charakterystyką jego użycia takimi jak: liczba możliwych wartości, rozkład wartości

(w przypadku innego typu pola niż kategoria liczba możliwych wartości jest trudna do określanie i mogła by wprowadzać w błąd, dlatego taki przypadek oznaczam wartością ‘-‘).

Dla ilustracji przedstawiłem ten profil dla wybranych atrybutów projektu MongoDB:

C12751|AtU : {Typ wyliczeniowy; 19,86%; 23 ; <0,37%; 0,00%; 2,42%; 2,91%; 1,16%; 1,49%; 2,63%; 2,74%; 2,06%; 1,16%; 0,86%; 0,27%; 0,25%; 0,01%; 0,15%; 0,03%; 0,66%; 0,33%; 0,14%; 0,16%; 0,01%; 0,01%; 0,03%; 0,01%>}

C17050|AtU : { Typ wyliczeniowy; 67,05%; 2 ; <64,36%; 2,69%>}

C16465|AtU: { Typ całkowity ; 34,59%; 191; < - >}

C10032|AtU: { Typ wyliczeniowy; 60,28%; 4; <60,15%; 0,06%, 0,04%, 0,02%>}

Widoczny jest niższy współczynnik wypełnienia atrybutów zgłoszenia o typie danych inny niż kategoria, oraz znaczący spadek tej miary w przypadku dużej liczby możliwych do wyboru wartości. Rozkład używanych wartości przedstawiony w profilu uwidacznia również dominację jednej wartości dla każdego typu kategorii. Relatywna wartość rozkładu wartości atrybutu *C17050* wynosi odpowiednio 64,36% (95,99%) dla pierwsza wartości oraz 2,69% (4,01%) dla drugiej wartości, gdzie w nawiasach podaję wartość bezwzględną. Podobnie cechy wykazuje rozkład wartości atrybutu *C10032* dla którego relatywna współczynnik procentowy(względem wszystkich zgłoszeń - 13434) wynosi odpowiednio *60,15% (99,79%)* dla pierwszej wartości, *0,06% (0,1%)* dla drugiej wartości, *0,04% (0,07%)* dla trzeciej wartości *oraz 0,02% (0,04%)* dla czwartej wartości. Dla atrybutów C10032, *C17050* najczęściej wybierana była pierwsza wartość spośród dostępnych, co może być związane np. z domyślną wartością pola. W takiej sytuacji, administrator projektu może przemyśleć i przedyskutować z zespołem istotność pola *C10032* oraz *C17050*. Bardziej równomierny rozkład używanych wartości atrybutu występuje w przypadku pola *C12751* gdzie bezwzględną miara dystrybucji (liczona względem zgłoszeń z polem *C12751*) dla kolejnych wartości pola jest równa: 1,87%; 0,00%; 12,18% ; 14,66%; 5,85%; 7,50%; 13,23%; 13,79%; 10,38%; 5,85%; 4,31%; 1,35%; 1,24%; 0,07%; 0,75%; 0,15%; 3,34%; 1,69%; 0,71%; 0,79%; 0,07%; 0,04%; 0,15%; 0,04%. Jedynie 5 wartości przekracza próg 10%, a aż 5 wartości charakteryzuje procent użycia wartości poniżej 1%. Taka obserwacja jest podstawą do przeprowadzenia oceny zasadności tak wielu opcji wartości (23) w polu kategorii *C12751*.

Profile *At|AtU* dla wybranych atrybutów projektu komercyjnego P1 wygląda następująco:

C10322|AtU : { Typ całkowity; 18,97%; 6; <1,68%; 4,54%; 3,15%; 5,09%; 4,12%; 0,38%>}

C12835|AtU : { Typ wyliczeniowy; 9,8%; 1; <9,8%>}

C12800|AtU : { Typ wyliczeniowy; 9,08%; 3; <8,96%; 0,04%; 0,08%>}

W projekcie komercyjnym P1 widoczne jest niskie wypełnienie pól dodatkowych, które nie są obowiązkowe. Oznacza to że aktorzy biznesowi nie przykładają staranności do rzetelnego raportowania dodatkowych parametrów zgłoszenia. Pole *C12835* posiada jedną wartość z małym stopniem wypełnienia (poniżej 10%) w skali wszystkich badanych zgłoszeń, jednak wartość ta przekracza 99% dla zgłoszeń typu *New Feature*. Istotne jest zatem przeprowadzanie bardziej szczegółowych analiz z uwzględnieniem typu zgłoszenia. Profil *At|AtU* może być wyznaczony również w takiej korelacji jako profil At|AtU|T według wzoru:

$$\text{At|AtU|T} = \{\text{typ atrybutu; liczba możliwych wartości; } <\textbf{procent wypełnienia atrybutu w określonym typie}; (\text{rozkład procentowy przyjmowanych wartości})>\} \quad (3.2)$$

gdzie At oznacza nazwę atrybutu, *AtU* charakterystykę typu wartości jako: typ danych, liczba możliwych wartości, stopień wypełnienia atrybutu oraz rozkład procentowy (relatywny, w odniesieniu do wszystkich zgłoszeń) występujących wartości w korelacji z typem zgłoszenia T przedstawionego jako sekwencję wartości. Dla projektów *open soruce* sekwencja ta zawiera typy zgłoszeń <*New Feature, Improvement, Bug*>. Dla ilustracji wyznaczyłem profil *At|AtU|T* projektu *open source* MongoDB dla wybranych pól:

C12751|AtU|T : {Typ wyliczeniowy; 23;

*<**32,08%**; (0,26%; 0,00%; 3,88%; 2,33%; 5,56%; 0,39%; 3,23%; 2,72%; 7,12%; 0,26%; 2,46%; 0,00%; 0,78%; 0,00%; 0,26%; 0,13%; 1,03%; 0,65%; 0,52%; 0,52%; 0,00%; 0,00%; 0,00%; 0,00%)>;*

*<**33,78%;** (0,09%; 0,00%; 4,39%; 5,66%; 1,15%; 2,84%; 4,67%; 5,43%; 3,88%; 1,62%; 1,43%; 0,26%; 0,49%; 0,05%; 0,14%; 0,02%; 1,01%; 0,28%; 0,14%; 0,12%; 0,00%; 0,02%; 0,07%; 0,00%)>;*

*<**11,68%**; (0,52%; 0,00%; 1,28%; 1,57%; 0,76%; 0,90%; 1,53%; 1,38%; 0,68%; 1,01%; 0,42%; 0,30%; 0,07%; 0,00%; 0,14%; 0,02%; 0,45%; 0,33%; 0,11%; 0,14%; 0,02%; 0,00%; 0,01%; 0,01%)>}*

*C17050|AtU|T : { Typ wyliczeniowy; 2; <**57,83%;** (53,04%; 4,79%)>; <**58,75%;** (54,73%; 4,02%)>; <**72,10%;** (70,28%; 1,82%)>*

*C10032|AtU|T : { Typ wyliczeniowy; 4; <->; <->; <**96,34%;** (96,13%; 0,10%, 0,07%, 0,04%)}*

Oraz dla wybranych pól projektu komercyjnego P1 gdzie sekwencja typów zgłoszeń odpowiada wartościom *<New Feature, User Story, Task, Bug>* :

*C10322|AtU|T : { Typ całkowity; 6; < - >; <**84,62%;** (8,5%; 16,19%; 15,79%; 27,94%; 14,98%; 1,21%)>; <**70,47%;** (5,56%; 19,59%; 10,53%; 15,20%; 17,84%; 1,75%)>; <**0,06%;** (0%, 0,06%, 0%, 0%, 0%, 0%>}*

*C12835|AtU : { Typ wyliczeniowy; 1; <**99,57%;** (99,57%)>; < - >; < - >; < - >}*

*C12800|AtU : { Typ wyliczeniowy; 3; <**3,42%;** (2,14%; 0,43%; 0,85%)>; <**83%;** (83%, 0%, 0%)>; <**0,88%;** (0,88%; 0%; 0%)>; < - >}*

W wartościach profilu *At|AtU|T* dla projektu *open source* MongoDB widoczne są niskie wartość procentowego wystąpienia poszczególnych wartości atrybutu *C12751* we wszystkich weryfikowanych typach zgłoszeń, ostatnie 13 wartości atrybutu niezależnie od typu zgłoszenia używanych jest w mniej niż 0,3% zgłoszeń. Analogiczna sytuacja występuje dla pozostałych profili *At|AtU|T* projektu MongoDB. Jeżeli dodatkowe pole jest ustawiane przez użytkownika najczęściej wybiera on pierwszą wartości z dostępnej listy. Przykładowo dla atrybutu *C17050* druga dostępna wartość atrybutu występuje odpowiednio dla 4,79% (*New Feature*), 4,02% (*Improvement*), 1,82% (*Bug*) zgłoszeń. Niski procent użycia wartości innej niż domyślna dla atrybutu, może świadczyć o braku rzetelności aktorów uczestniczących w procesie obsługi zgłoszeń. Którzy nie zwracają uwagi na wartości pól z domyślnymi wartościami, wskazuje to na pewne niedoskonałości raportowania zgłoszeń w projekcie *open source* MongoDB. Profil korelacji rozkładu wartości atrybutu z typem zgłoszenia dla projektu komercyjnego P1 np. *C10322|AtU|T* wykazuje równomierny *rozkład wartości,* podobnie w przypadku atrybutu *C12800.* W tym wypadku jednak równomierny rozkład występuje jedynie dla typu zgłoszenia *New Feature,* dla typu *User Story* widoczna jest dominacja pierwszej wartości (83%). Sytuacja ta związana jest bezpośrednio ze specyfiką pola, które wskazuje na zatwierdzanie zmiany przez właściciela produktu. Dla zgłoszeń *User Story* są one zawsze potwierdzane przed zgłoszenia

dodaniem do sprintu, stąd występująca dominacja pierwszej wartości która oznacza pozytywną weryfikację właściciela produktu. Szczegółowa interpretacja rozkładu wartości atrybutów względem typu zgłoszenia w projekcie P1, jest możliwa z uwagi na moje bezpośrednie zaangażowanie w projekt, czego nie mogłem wykonać w projektach *open soruce*. Potwierdza to wcześniejsze założenia że szczegółowa interpretacja takich profili powinna być wykonywana przez ekspertów danego projektu.

Analiza stopnia wypełnienia atrybutów zgłoszenia w korelacji z typem pola przedstawia szczegółowy (drobnoziarnisty) obraz konfiguracji repozytorium ITS. Pozwala też na ocenę użycia pól oraz ich poszczególnych wartości w procesie obsługi zgłoszeń. Takie weryfikacje są często pomijane w projektach, przez co w repozytoriach występują pola o niskim stopniu użycia lub z jedną dominującą wartością (np. z wykorzystaniem powyżej 90%). Może być to spowodowane niezrozumieniem ich nazwy przez aktorów procesu, ich zaniedbaniem lub pomijalnej wartości informacyjnej pola. Zasadne zatem jest przeprowadzanie ewaluacji od ogółu do szczegółu, z uwzględnieniem korelacji typu zgłoszenia. Wnioski z nich płynące mogą stanowić podstawę do poprawienia konfiguracji atrybutów zgłoszenia lub wyjaśnienia ich poszczególnych wartości aktorom procesu. Więcej szczegółowych korelacji pomiędzy atrybutami zgłoszeń przedstawiłem w sekcji 3.3.4.

3.3.2. Analiza aktywności aktorów

Niektóre kwestie raportów zgłoszeń mogą interesować w szczególny sposób kierowników (*manager-ów*) projektów, co wymaga głębszych badań. Jednym z takich obszarów jest analiza aktywności użytkowników. Ogólne statystyki projektów uwzględniają liczbę aktorów oraz liczbę zgłoszeń, które raportują. Liczba reporterów oraz ich zaangażowanie mogą się różnić. Musimy brać pod uwagę fakt, że jedni aktorzy mogą być zaangażowani bardziej niż inni. Typowe jest, że mała grupa interesariuszy projektu dominuje prowadzone prace. Dlatego, wartościowe jest wprowadzenie pewnych profili agregacyjnych raportów, reprezentujących liczbę reporterów dodających określony procent zgłoszeń, np. 50%, 80%, 100%. Są to tak zwane profile całościowe. W wielu projektach możemy zaobserwować dużą rotację zespołu, stąd wartościowe jest ujęcie w takich profilach perspektywy okresów czasowych.

Przykładem takiego raportu jest Tab. 6, w której przedstawiono perspektywę kolejnych 13 lat dla 2 projektów. Ostatni wiersz tabeli prezentuje profil całościowy (sumaryczny) dla całego okresu pokazanego w tabeli. Tab. 6 została opracowana na podstawie zagregowanych danych przedstawionych w [84] dla projektów Apache Casandra (Casandra)

oraz Apache Spark (Spark). Dla obydwu projektów niezależnie od roku 50% zgłoszeń tworzonych jest przez małą grupę użytkowników. W projekcie komercyjnym P1, widoczna jest pewna analogia tej obserwacji prezentowana w Tab. 8, gdzie 64,9% zgłoszeń tworzonych jest przez 9 testerów.

Rok Projektu	Casandra 13 lat			Spark 12 lat		
	50%	80%	100%	50%	80%	100%
1	4	16	99	1	2	4
2	9	51	245	2	7	18
5	29	151	336	24	175	911
6	29	174	533	25	282	1473
12	11	57	198	20	145	828
13	10	37	176	15	82	295
Łącznie dla 13 lat	51	429	2917	64	940	6752

Tab. 6 Zagregowane profile aktywności raportowania zgłoszeń aktorów projektów

Profile można generować w perspektywie ogólnej obejmującej wszystkich aktorów i typy zgłoszeń lub w perspektywach przedefiniowanych (przekrojowych), obejmujących tylko błędy, tylko deweloperów, określone priorytety zgłoszeń, itp. Poniżej podano ich ilustracje.

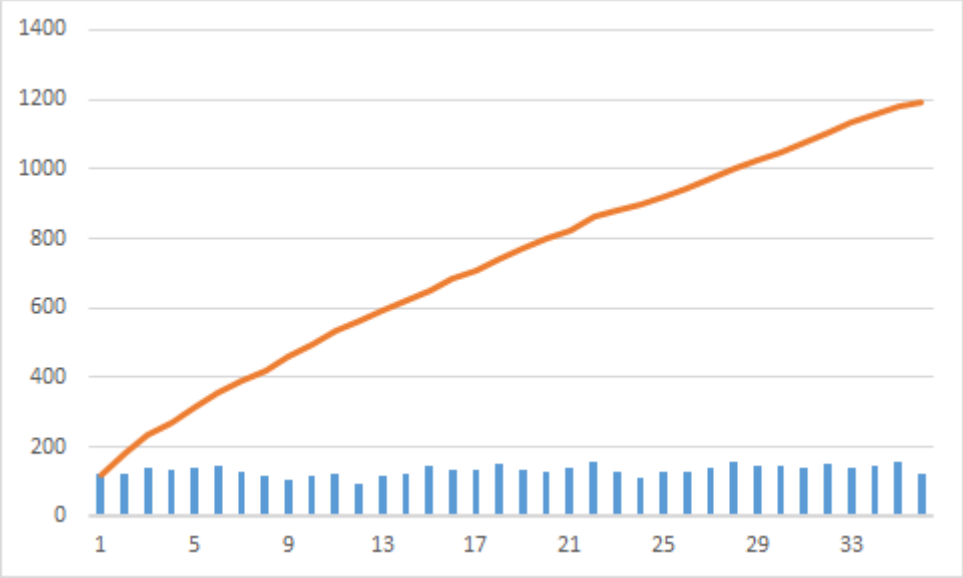
Metryką aktywności reporterów jest czas ich zaangażowania w projekt. Jest to szczególnie istotne w długich projektach oraz okresach fluktuacji uczestników projektu. Metryka może przyjąć dwie postaci: 1) absolutną – biorąca pod uwagę datę pierwszego i ostatniego zgłoszenia dodanego przez reportera oraz 2) agregacyjną – sumującą okresy czasowe poszczególnych aktorów, w których określamy okres jako czas (DT) między dwoma następującymi po sobie aktywnościami dodania nowego zgłoszenia. Jeżeli czas ten jest większy niż zadany DT, rozpoczynamy naliczanie kolejnego okresu. Przykład takiej agregacji ilustruje Tab. 7, gdzie przedstawione parametry czasowe (dni) obejmują medianę (Med), kwartyl Q3, wartość maksymalną (Max) i średnią (Av) dla dwóch wariantów okresu DT: a = 4 tygodnie, b=52 tygodnie.

Param	cas	spark	flink	ignite	moz	red
Med - a	2.0	2.0	3.0	2.0	1.0	4.0
Med - b	3.0	2.0	4.0	4.0	1.0	29.0
Q3 – a	15.0	9.0	22.0	28.0	2.0	34.0
Q3 - b	85.0	60.0	70.0	161.0	3.0	410.5
Max – a	3501	3094	2809	2399	6451	1871
Max - b	4474	3605	2937	2646	8229	3842
Av – a	45.7	26.5	47.9	114.5	11.4	62.8
Av - b	142.2	104.4	137.9	209.3	76.9	302.1

Tab. 7 Agregacja metryki aktywności czasowej reporterów

Tab. 7 opracowana została na podstawie danych z pracy [84] dla projektów Apache Casandra (cas), Apache Spark (spark), Apache Flink (flink), Apache Ignite (ignite), Mozilla Tunderbird (moz), Red Hat Enterprise Linux 8 (red).

Interesującą perspektywą aktywności aktorów jest rozkład ich zaangażowania w czasie, którą przedstawiłem na Rys. 1 wraz z agregacją całkowitej liczby reporterów. Miara ta ilustruje również poziom fluktuacji w projektach. Wysoki poziom fluktuacji aktorów w projekcie ma duży wpływ na szereg trafności predykcji (np. w modelach *SRGM*). Niestety jest to pomijane w publikacjach bazujących na analizie gruboziarnistej. Wykres dotyczy projektu MongoDB dla zgłoszeń typu *Improvement*, *New Feature*, *Bug* z łączną liczbą zgłoszeń 13434. W projekcie widoczna jest wysoka fluktuacja reporterów zgłoszeń, z uwagi na liniowy przyrost wartości zagregowanej (linia wykresu), oraz równego histogramu określającego liczbę reporterów w danym miesiącu.



Rys. 1 Wykres liczby aktywnych reporterów projektu MongDB

Bardziej szczegółowa analiza powinna uwzględniać rolę reporterów oraz osób komentujących zgłoszenia w czasie ich obsługi. Podział na poszczególne role aktorów jest zazwyczaj sformalizowany w projektach komercyjnych (moje bezpośrednie zaangażowanie w projekt pozwala na przydzielenie takich ról do reporterów). Dane te są jednak niedostępne dla projektów *open source*. JIRA pozwala na przypisanie użytkowników do grup ról reprezentujących różne odpowiedzialności (np. programista, tester). Informacja ta wspomogłaby opracowanie takiej perspektywy, jednak zweryfikowane projekty *open source* były pozbawione takiej konfiguracji lub dostęp do niej był ograniczony. W projekcie P1 wyróżniono 6 ról: Tester (9), Analityk (6), Właściciel produktu (1), Projektant interfejsów użytkownika (1), Programista (15), Architekt (2). W nawiasach podałem liczbę aktorów powiązanych z profilem. Szczegółowy obraz dystrybucji tworzonych zgłoszeń między priorytetami przedstawia Tab. 8. Tab. 9 przedstawia dystrybucję typów tworzonych zgłoszeń względem profilu reportera. Tabele zawierają współczynnik procentowy liczby tworzonych zgłoszeń powiązanego profilu względem liczby wszystkich zgłoszeń oraz liczbę zaangażowanych aktorów danego profilu po znaku „/”. Kolumny przedstawiają odpowiednio priorytet zgłoszenia (Tab. 8) oraz typ zgłoszenia (Tab. 9). Ostatnia kolumna reprezentuje sumę współczynnika procentowego wszystkich stworzonych zgłoszeń przez powiązaną grupę profili. Tab. 8, Tab. 9 agregują 2378 zgłoszeń projektu komercyjnego P1: 234 (*New Feature*), 247 (*User Story*), 342 (*Task*) i 1555 (*Bug*).

Profil	Blocker	Critical	Major	Minor	Low	Suma
Tester	1,05% /2	9,88% /7	25,53% /9	19,72% /8	1,56% /6	57,74%
Programista	0,04% /1	0,08% /1	0,17% /2	1,85% /4	0%	2,14%
Architekt	0,21% /1	0,55% /2	0,59% /2	0,97% /2	0%	2,31%
Analityk	0,08% /1	3,15% /4	6,85% /4	25,90% /5	0%	36,00%
Właściciel produktu	0,34% /1	0,55% /1	0,42% /1	0,17% /1	0%	1,47%
Projektant interfejsu użytkownika	0%	0%	0,04% /1	0,25% /1	0,04% /1	0,34%

Tab. 8 Zagregowane profile grupowe aktywności reporterów tworzonych zgłoszeń dla projektu P1 (priorytet).

Profil	New Feature	User Story	Task	Bug	Suma
Tester	0,34% /1	0,71% /2	1,01% /3	55,68% /9	57,74%
Programista	0%	0%	0,29% /3	1,85% /4	2,14%
Architekt	0,04% /1	0,04% /1	1,64% /2	0,59% /2	2,31%
Analityk	8,12% /4	9,63% /4	11,35% /5	6,90% /5	36,00%
Właściciel produktu	1,26% /1	0%	0,04% /1	0,17% /1	1,47%
Projektant interfejsu użytkownika	0,08% /1	0%	0,04% /1	0,21% /1	0,34%

Tab. 9 Zagregowane profile grupowe aktywności reporterów tworzonych zgłoszeń dla projektu P1 (typ zgłoszenia).

Szczegółowa dystrybucja profili względem priorytetów oraz typów tworzonych zgłoszeń wskazuje na dwie główne cechy projektu:

1) Profile zaangażowania aktorów są na bardzo dobrym poziomie, tzn. osoby znają swoje role i skupiają się na swoich codziennych zadaniach. Architekci odpowiadają za bardzo mały procent tworzonych zgłoszeń (2,31%). Główną rolą Architekta jest doradztwo i uczestniczenie w projektowaniu nowych rozwiązań. Na co dzień sprawdzają oni nowe funkcjonalności opisane przez Analityków, Właściciela Produktu lub przekazują im opis techniczny rozwiązania. Odzwierciedla się to w małej aktywności tworzenia zgłoszeń w repozytorium JIRA. Główna dystrybucja zgłoszeń opiera się o dwa profile: Tester oraz Analityk, co jest zgodne z założeniami ich odpowiedzialności. Podobne obserwacje dotyczą profilu Programisty oraz Projektanta interfejsów użytkownika, którzy w głównym stopniu odpowiadają za rozwiązanie zgłoszenia (Programista) i dostarczeniu projektów interfejsów użytkownika, nowych funkcjonalności (Projektant interfejsu użytkownika).

2) Wśród zespołu testerów występuje mała fluktuacja, jednak dystrybucja ich zaangażowania w tworzenie zgłoszeń wydaje się odpowiednia bez znacznego skupienia w jednym priorytecie. Stabilność zespołu projektu komercyjnego P1 znacząco różni się od specyfiki projektów typu *open source* w których fluktuacja zespołu jest wysoka co zostało przedstawione w Tab. 6.

Istotnym elementem pogłębienia szczegółowych analiz (drobnoziarnistych) jest sprawdzenie aktywności aktorów projektowych w kolejnych fazach procesu obsługi zgłoszeń, w szczególności specyfiki dodawanych komentarzy, dyskutowanych w sekcji 3.3.3. oraz 4.3.2. Profile aktywności aktorów mogą być także wyznaczone w korelacji z typem zgłoszenia, takie analizy uwzględniłem w sekcji 3.3.4.

3.3.3. Profile komentarzy zgłoszeń

Podstawową cechą systemów ITS jest możliwość dodawania komentarzy do zgłoszeń. Na podstawie tej funkcjonalności definiuję miernik aktywności aktorów oraz cechę postępu rozwiązania zgłoszenia. Rozważam 5 charakterystyk: 1) rozkład liczby komentarzy względem raportowanych zgłoszeń, 2) rozkład liczby aktorów komentujących zgłoszenie, 3) dystrybucję długości komentarza (np. jako liczba słów w komentarzu), 4) rozkład czasu trwania sekwencji komentarzy, 5) dystrybucję liczby komentarzy względem liczby zmian w kodzie.

Tab. 10 opracowana została na podstawie danych z pracy [84] dla projektów Apache Casandra (cas), Apache Spark (spark), Apache Flink (flink), Apache Ignite (ignite), Mozilla Tunderbird (moz), Red Hat Enterprise Linux 8 (red). Przedstawia połączenie charakterystyk średniej liczby komentarzy wraz ze średnią liczbą osób dodających komentarze oraz czasu sekwencji komentarzy w analizowanych projektach. Mediana liczby komentarzy względem 3 typów zgłoszeń, była w zakresie odpowiednio, 2-4, 2-3 , 2-47 komentarzy. Jednakże dystrybucja wartości maksymalnych była znacząco wyższa, odpowiednio: 126-506 komentarzy, 13-107 komentarzy, 2275-7488 dni. Wielkość komentarzy była z zakresu: mediana: 10-39, trzeci kwartyl: 32-69 oraz wartości maksymalne: 4834-107000.

Parametr	cas	spark	flink	ignite	moz	red
Med – a	4/3	2/2	3/2	2/2	4/3	2/2
Med - b	5.3	1.9	4.2	2.3	16.2	47.5
Q3 – a	8/3	4/3	7/3	4/3	7/4	4/3
Q3 -b	48.8	24.0	63.7	21.7	300.0	201.1
Max – a	295/29	126/43	370/22	144/13	506/107	161/42
Max - b	3619.6	3195.6	2803.4	2275.7	7488.4	6152.7
Av – a	6.9/2.9	3.3/2.3	6.0/22.0	3.2/2.4	6.9/3.2	3.2/2.4
Av - b	100.9	73.3	123.6	55.0	360.6	159.3

Tab. 10 Statystyka komentarzy: a – liczba komentarzy / liczba komentujących w zgłoszeniu, b - czas sekwencji komentarzy w dniach

W każdym analizowanym przeze mnie projekcie widoczna jest grupa najbardziej aktywnych użytkowników pod względem dodawanych komentarzy. W badanym projekcie MongoDB duża liczba komentarzy dodawana jest przez użytkownika *Githook user* (automat), który odpowiada za tworzenie około 600-800 komentarzy na kwartał, gdzie najbardziej aktywni użytkownicy dodają między 5 a 70 komentarzy w tym samym zakresie czasu. Komentarze dodawane przez konto *Githook user*, są wyzwalane przez specjalne zdarzenia w repozytorium kodu. Głównym celem działania jest automatyzacja niektórych części procesu dostarczania

rozwiązania w ramach ciągłej integracji (CI, ang. *Continues Integration* - proces sprawdzający nowo dodany kod przed włączeniem (ang. *merge*) go do docelowej wspólnej gałęzi kodu). Zastosowanie automatu dodającego komentarze poprawia przejrzystość informacji oraz zapewnia ich ciągłość. W przypadku braku powiązania między repozytorium kodu, a systemem ITS, odpowiedzialność za dodanie informacji o zmianie w kodzie (link do *pull request*) przesunięta jest na programistę. W projekcie komercyjnym, gdzie nie jest stosowany skrypt komentujący zgłoszenia, sporadycznie widoczne są braki tych informacji. Podsumowując, automatyzacja zapewnia ciągłość procesu wymiany danych pomiędzy ITS, a repozytorium kodu. Niemniej widoczna jest potrzeba bardziej wysublimowanych analiz dostępnych danych testowych komentarzy, np. poprzez metody eksploracji tekstu oraz klasyfikacji komentarzy do określonych klas. Temat ten omówiony jest w rozdziale 4. pracy.

3.3.4. Analiza korelacji atrybutów zgłoszeń

Zaprezentowane modele rozkładu statystycznych cech repozytoriów ITS mogą być wyznaczone niezależnie lub z uwzględnieniem zależności od innych pól lub innych połączonych repozytoriów. Standardowo możemy uzależnić zagregowane dane od filtrów, takich jak np. typ zgłoszenia, priorytet, istotność, rola reportera. Dostęp do tych danych opiera się o specyficzne pola zgłoszeń, jednak może nie być oczywisty. Opisany wcześniej problem z brakiem ról użytkowników w projektach typu *open source*, utrudnia te analizy, jednak możliwe jest przybliżone określenie roli użytkownika na podstawie obserwacji jego aktywności w repozytorium, przykładowo: zaangażowanie w tworzenie zgłoszeń, specyfika dodawanych komentarzy bądź zmian w kodzie. Można do tego celu wykorzystać techniki analizy tekstowej (ang. *text mining*) rozpatrywane w rozdziale 4. Dodatkowym kierunkiem badań statystycznych są analizy korelacyjne. Wyróżniłem tu kilka podstawowych: 1) korelacje między atrybutami zgłoszeń, 2) korelacje czasowe zgłoszeń, 3) korelacje zgłoszeń ze zmianami kodu. Wygodną formą ich prezentacji są wektory korelacji, których przykłady przedstawiłem poniżej.

Zależności między atrybutami zgłoszeń można przedstawić w postaci tablic korelacji (metryk) między wartościami dwóch atrybutów A (z n wartościami kategorii) i B (z m wartościami kategorii). Określam zgodnie ze schematem:

$$A|B: \{< a1|b1 >, < a2|b2 > \ldots < am|bm >\} \qquad (3.3)$$

gdzie a_i $(1 \leq i \leq m)$ to wektory określające liczbę spraw w ramach kolejnych kategorii atrybutu A, które zakładają jednocześnie wartości b_i atrybutu B.

Dla ilustracji przedstawiłem tablicę korelacji dla czterech par atrybutów, projektu Apache Casandra (opracowane na podstawie danych z [84]): *Ch|P, Ch|T, Co|P i Co|T,*

które odpowiadają atrybutom: kategoria zmiany (Ch – <*semantic, code clarity, quality, performance, operability*>), priorytet (P - {*low, normal, high, urgent*}), typ zgłoszenia (T – {*Improvement, New Feature, Task, Bug*}), złożoność (Co – <*low, normal, challenging, byzantine*>), wartości wektorów pierwszego atrybutu podane są w nawiasach <…>, podsekwencje w nawiasach {…} odpowiadają wartościom drugiego atrybutu:

Ch|P: {<11, 14, 31, 9, 36>,<182, 79, 200, 64, 310>,<0, 3, 7, 1, 13>,<8, 0, 1, 2, 0>}
Ch|T: {<63, 37, 126, 58, 229>,<10, 2, 8, 2, 55>,<123; 56, 103, 13, 70>,
<2, 0, 1, 2, 4>}
Co|P: {<152, 40, 5, 0>,<762, 1202, 65, 4>,<9; 15, 5, 0>,<8, 22, 23, 0>}
Co|T: {<324, 249, 20, 0>,<19, 57, 7, 0>,<137; 238, 9, 0>,<0, 2, 2, 1>}

Atrybut Ch uzupełniony jest głównie dla zgłoszeń typu *Improvement, New Feature i Task*, które wymagają pewnych zmian (w przeciwieństwie do błędów ang. *Bug*). Atrybut złożoności wypełniony jest głównie w przypadku zgłoszeń o priorytecie *normal i low*. Dla wyższych priorytetów zgłoszeń informacja ta uznawana jest za nieprzydatną z uwagi na potrzebę szybkiego rozwiązania niezależnie od złożoności zgłoszenia. Należy zauważyć również że stopień wypełnienia porównywanych atrybutów może się różnić. W rozpatrywanych profilach atrybuty priorytet i typ są wypełnione w 100%, natomiast kategoria zmiany (Ch) i złożoność (Co) odpowiednio 5,2% oraz 13,1%. Rozkład wypełnienia współczynnika kategorii zmiany w odniesieniu do priorytetu wynosił $ChF|P$ = {0,10, 0,87, 0,02, 0,01}, a w odniesieniu do typu zgłoszenia $ChF|T$ = {0,08, 0,88, 0,01, 0,02} i $CoF|T$ = {0,26, 0,04, 0,16, 0,54, 0,00}. Interpretacja profili korelacji w przypadku atrybutów o współczynniku wypełnienia 100% jest łatwiejsza. W szczególności skorelowanie reporterów z innymi atrybutami wskazuje ich odpowiedzialność w dokumentacji. Jest to przydatne w projektach z udziałem wielu reporterów (często zmieniających się). Takie profile wyznaczyłem dla projektu komercyjnego P1. Zbadałem w nich korelację grupy reporterów (GR - <testerzy, programiści, architekci, analitycy, właściciele produktu, projektanci interfejsu użytkownika>) z typem zgłoszenia (T – {*New Feature, Improvement* (odpowiada zadaniu *User Story*), *Task, Bug}*) oraz priorytetem (P – *{Blocker, Critical, Majro, Minor, Trivial}*), wartości wektorów pierwszego atrybutu podane są w nawiasach <…>, podsekwencje w nawiasach {…}:

GR|T: {<8, 0, 1, 193, 30, 2>, <17, 0, 1, 229, 0, 0>, <24, 7, 39, 270, 1, 1>, <1324, 44, 14, 164, 4, 5>}
GR|P: {<25, 1, 5, 2, 8, 0>, <235, 2, 13, 75, 13, 0>, <607, 4, 14, 163, 10, 1>, <469, 44, 23, 616, 4, 6>, <37, 0, 0, 0, 0, 1>}

Zauważalna jest duża korelacja między typem grupy użytkowników w projekcie komercyjnym P1 a typem tworzonych zadań *GR|T*, wynika to z jasnego podziału ról w zespole. Najwięcej zgłoszeń błędów tworzonych jest przez grupę testerów, natomiast zgłoszenia typu *New Feature, Improvement* zdominowane są przez reporterów z grupy analitycy i właściciele produktu. Widoczna jest także duża różnorodność priorytetów zgłoszeń dodawanych przez grupę testerów, co odróżnia profil ich zgłoszeń od innych grup w których priorytet zgłoszenia nie pełni kluczowej roli. W przypadku błędów pole priorytet pełni kluczową rolę determinującą czas rozwiązania zgłoszenia. Pola Priorytet oraz Typ zadania w projekcie komercyjnym są wypełnione w 100%. W przypadku niskiego współczynnika wypełnienia porównywanych atrybutów korelacje mogą nie być łatwe w interpretacji. Głębsza analiza może dotyczyć profili warunkowych z dodatkowymi atrybutami które poprawiają względny współczynnik wypełnienia porównywanych atrybutów. Przedstawione profile korelacji ilustrują relacje między atrybutami i zgłoszeniami. Podobnie można wprowadzić profile korelacji obejmujące różne repozytoria np. repozytorium zgłoszeń i kodu.

Każda z generowanych statystyk jest określana w danym momencie czasowym, np. do konkretnego dnia bądź w wybranych zakresach czasowych. Obserwacje te mogą być często zmienne szczególnie w projektach o dużej intensywności aktywności (np. *MongoDB*: 377; 388 – liczba nowych zgłoszeń odpowiednio w styczniu i lutym 2023). Istotne jest zatem, aby szczegółowe analizy opierały się także na obserwacji trendów w następujących po sobie okresach. Co więcej, możemy ocenić w ten sposób poprawę bądź pogorszenie jakości projektu, np.: oceniając liczbę dystrybucji priorytetów zgłoszeń, aktywności aktorów, czasu obsługi zgłoszeń itp. Analizy te możemy prowadzić na podstawie dwóch klas obserwacji: przyrostowej i kumulatywnej.

Korelację zgłoszeń ze zmianami w kodzie definiuję jako wektor rozkładu liczby zgłoszeń względem liczby wymaganych poprawek niezbędnych do zamknięcia zgłoszenia. Wektor opisany jest przez wartości *<liczba zgłoszeń błędów; liczba zgłoszeń błędów wymagających poprawki; procent zgłoszeń rozwiązanych przez 1 poprawkę, procent zgłoszeń rozwiązanych przez 2-4 poprawek, procent zgłoszeń rozwiązanych przez więcej niż 4 poprawki*>. Dla zobrazowania wyznaczyłem wektory dla zgłoszeń błędów w 3 następujących po sobie kwartałach projektu MongoDB, przyjęły one odpowiednio wartości: *<477; 344; 48,55%; 47,09%; 4,36%>,<693; 477; 64,36%; 32,29%; 3,35%>, <522; 312; 51,28%; 46,15%; 2,56%;>* Wektor może zostać wyznaczony również dla całego projektu lub innego typu zgłoszenia.

Wartościowe jest przeprowadzanie szczegółowej analizy dla konkretnych typów zgłoszeń, ponieważ wektor ogólny (dla całego projektu) daje jedynie gruboziarnisty obraz. W analizowanym oknie czasowym widoczna jest niezasadność wielu błędów, bądź są one rozwiązywane bez zmian w kodzie (różnica między liczbą wszystkich zgłoszeń, a zgłoszeń rozwiązanych przez zmianę w kodzie). Kwestia ta wymagałaby indywidualnej oceny zgłoszeń (bez zmian w kodzie). Stosunkowo duży procent zgłoszeń – powyżej 2,5% w każdym kwartale – wymagał więcej niż 4 poprawek. Zaobserwowałem również zgłoszenia, których rozwiązanie zostało osiągnięte dopiero po 8-ej poprawce. Sytuacja ta nie ma miejsca w projekcie komercyjnym, w którym największa liczba wymaganych poprawek dla rozwiązania błędu wynosi 3. W każdym z badanych okresów projektu komercyjnego ponad 87% zgłoszeń rozwiązywanych było przez 1 poprawkę. Widoczna jest także możliwość prowadzenia kolejnych badań przy użyciu eksploracji tekstów dla uzasadnienia poprawności zgłoszeń błędów, które nie wymagały zmian w kodzie. Szczegółowe badania zmian w kodzie oraz korelacji ze zgłoszeniami nie jest przedmiotem mojej rozprawy, jednak były one prowadzone w Instytucie np. w pracy [82].

Badanie głębszych korelacji atrybutów zgłoszeń wymaga zastosowania zaawansowanych metod eksploracji tekstowej opisów zgłoszeń i komentarzy (*text miningu*) lub informacji zewnętrznych (np. kompetencji i doświadczeniu aktorów) i kontekstu. Dyskusję, dotyczącą metod analizy tekstów (*text mining)* w repozytoriach zgłoszeń, przedstawiłem w rozdziale 4. pracy. Eksploracja korelacji atrybutów zgłoszeń może zostać również połączone z procesem obsługi zgłoszeń i ścieżek modelowanych w grafach IHG, które omówiłem w sekcji 5.

4. Analiza tekstowa raportów zgłoszeń

Zgłoszenia raportowane w repozytoriach ITS zawierają wiele pól ukierunkowanych na poszczególne cechy informacyjne, określające między innymi: reportera, priorytet, istotność, typ zgłoszenia, wersję, której dotyczy zgłoszenie itp. Jednak najwięcej informacji przechowują pola opisowe, takie jak: tytuł, opis oraz komentarze zgłoszenia. W tych polach możemy wykorzystać techniki eksploracji tekstu (*text mining*), aby uzyskać większą wiedzę. Może być ona pomocna między innymi we wspomaganiu zarządzania projektami, pewnej ocenie efektywności procesu obsługi błędów a w szczególności klasyfikacji zgłoszeń.

W ostatnich latach widoczna jest zwiększona popularność prac w obszarze pozyskiwania wiedzy z danych, w szczególności danych tekstowych [85]. Informacje te mogą być wykorzystywane w wielu obszarach, np. śledzenie trendów rynkowych/informacyjnych poprzez analizę mediów społecznościowych (*twitter*), wykrywanie niechcianych treści, analiza wyszukiwanych fraz przez użytkowników wyszukiwarek bądź przetwarzanie dużych zbiorów danych w celu uzyskania pewnej wiedzy naukowej.

Wśród zadań eksploracji danych możemy wyróżnić 4 główne grupy [86]:

1) Pozyskiwanie informacji (ang. *information retrival*) – obszar zajmujący się przetwarzaniem tekstu w celu ekstrakcji konkretnych informacji, jak np. informacji o osobach, miejscach oraz możliwych połączeniach między danymi.
2) Klasyfikacji danych (ang. *classification*) – metody eksploracji danych pozwalające na przypisanie zadanego dokumentu (tekstu) do wcześniej określonych grup na zasadzie podobieństwa. Metoda ta należy do zbioru mechanizmów uczenia maszynowego z nadzorem (ang. *supervised learning*), z uwagi na potrzebę wcześniejszego zbudowania modelu klasyfikacji, poprzez określenie modeli podobieństwa i grup do jakich klasyfikowane są dokumenty.
3) Grupowanie (ang. *clustering*) – metody *text miningu*, których celem jest przeprowadzenie automatycznego grupowania zadanych dokumentów według podobieństwa. Różnica między grupowaniem a klasyfikacją opiera się o uczenie z nadzorem i bez. W klasyfikacji dokument przypisywany jest do wcześniej określonych grup, w grupowaniu natomiast grupy tworzone są automatycznie.
4) Podsumowanie tekstu (ang. *text summarization*) – metody, których celem jest automatyczne wygenerowanie podsumowania zadanego tekstu, poprzez wyznaczenie najważniejszych zdań. Szczególnym przykładem takich analiz może być generowanie słów kluczowych [87, 88].

Dla wsparcia tych zadań opracowano wiele algorytmów na podstawie przetwarzania języka naturalnego (NLP, ang. *Natural Language Processing*), które uwzględniają ekstrakcję informacji, analizę semantyczną, statystyczną oraz leksykalną, eksplorację danych i ich wizualizację.

Wykorzystanie klasycznych technik eksploracji tekstu odpowiednich do przetwarzania języka naturalnego na potrzeby publikacji, analizy wymiany krótkich wiadomości w mediach społecznościowych, np. *twitter* itp. jest niewystarczające i niezadowalające w przypadku repozytoriów ITS. Wynika to głównie z: żargonu technicznego, skrótów, łączenia zwrotów języka naturalnego z nazwami technicznymi niezgodnymi ze słownikiem thesaurus, wycinkami kodu, adresami email, odnośnikami do stron internetowych, specjalnych skryptów, wartości numerycznych, które pojawiają się w opisach zgłoszeń bądź komentarzy dodawanych przez aktorów projektu. Wyrażenia te znacząco zaciemniają zrozumienie sensu tekstu. Ponadto są one pomijane w dostępnej literaturze.

Opracowane schematy eksploracji tekstu zostały wzbogacone o głęboki wgląd w używany słownik składający się z wyrażeń słownika tezaurus (NL) oraz innych fraz. Słowa nienależące do słownika tezaurus są rozpoznawane i klasyfikowane zgodnie z ich znaczeniem semantycznym. Dodatkowo, autorski model klasyfikacji wsparty jest różnymi cechami statystycznymi analizowanego tekstu. Wszystkie wymienione cechy (semantyczne i statystyczne) poszerzają zakres eksploracji tekstu, co wyróżnia pracę na tle klasycznej eksploracji tekstu, opierającej się jedynie na słowniku języka naturalnego. Głównym celem przedstawionej analizy jest klasyfikacja ukierunkowana na specyficzne aspekty obserwacji, których celem jest wsparcie rozwoju projektu. Istotnym atutem opisanej analizy jest ewaluacja jakości raportowanych tekstów i sugestie potencjalnych usprawnień.

Rozdział 4. składa się z 4 sekcji. Pierwsza poświęcona jest szczegółowej analizie dostępnych badań z powiązanego zakresu literatury. Druga sekcja prezentuje opracowane oryginalne analizy cech tekstów, natomiast trzecia przedstawia szczegóły autorskiego algorytmu klasyfikacji z przykładowymi wynikami. Całość rozważań podsumowana została w dyskusji omówionej w sekcji 4.4.

4.1. Analiza literatury

Proces wytwarzania oprogramowania wspierany jest przez wiele rozwiązań, między innymi systemy śledzenia zgłoszeń (ITS) oraz systemy kontroli wersji (SVC). Zawartość tych repozytoriów zależna jest od przyjętych zasad dokumentacji w organizacji bądź zespole projektowym. Stanowią one bogate źródło informacji dla programistów, testerów, analityków

oraz innych interesariuszy. Tworzone zgłoszenia odzwierciedlają błędy, specyfikacje, wnioski lub sugestie nowych funkcjonalności, itp. Manualna weryfikacja takich repozytoriów jest żmudna
i czasochłonna, dlatego też w literaturze proponowane są różne podejścia automatyzacji ukierunkowane na różne aspekty ułatwiające eksplorację treści zgłoszeń.

W [72] autorzy podjęli problem zatwierdzania bądź odrzucania zgłoszeń do dalszego procesowania. Wykorzystują techniki przetwarzania języka naturalnego z oceną sentymentu (pozytywny, negatywny) opisu zgłoszenia, bazując na często występujących słowach. W [49] zgłoszenia są klasyfikowane do 3 kategorii: błąd, rozbudowa, pytanie. Autorzy wykorzystują do tego sieci neuronową *RoBERT* dostosowaną do raportów zgłoszeń.

W większości przypadków zgłoszenia wzbogacone są różnymi komentarzami, dodawanymi w procesie obsługi zgłoszeń przez aktorów projektu. Informacje te są użyteczne w procesowaniu zgłoszenia, monitorowaniu ogólnego postępu projektu, weryfikacji aktywności i kompetencji uczestników projektu itp. W [74] zaproponowana została technika pozyskiwania ciekawych informacji od interesariuszy projektów. Autorzy śledząc zgłoszenia projektów GitHub z dużą liczbą komentarzy, dodawanych przez dużą liczbę aktorów (średnio 10), sformułowali 15 kategorii występujących dyskusji, m. in. oczekiwane i zaistniałe zachowanie aplikacji, odtworzenie błędu, dyskusja rozwiązania, testowanie, rozmowy, postęp zgłoszenia, nowe zgłoszenie itp. Dyskusja ta daje jednak tylko ogólny wgląd w komentarze konkretnego zgłoszenia.

Autorzy [50] analizują możliwość klasyfikacji zgłoszeń jako błąd lub nie błąd, bazując na wytrenowanym modelu analizy tytułu i opisu zgłoszenia. Problem ten wybrany został na podstawie obserwacji repozytoriów, w których występowały zgłoszenia z opisem niepasującym do raportowanego typu zgłoszenia. W [89] wykorzystywane są metody eksploracji tekstu do etykietowania błędów, będących błędami bezpieczeństwa. Metody analizy tekstów (*text mining)* do określania sentymentu komentarzy przedstawiono w [90]. Szczegółowa dyskusja dotycząca precyzji istotności zgłoszenia zawarta została w [80]. Większość z opisanych metod wykorzystuje nieustrukturyzowane cechy tekstu, uczenie maszynowe oraz techniki eksploracji tekstu. W [91] zaproponowano narzędzie do precyzyjnej lokalizacji błędu na poziomie metody, przy wykorzystaniu podobieństwa semantycznego, współczynnika zależności i bliskości czasowej.

Eksploracja tekstu jest użyteczna również w rozwiązywaniu specyficznych problemów procesu obsługi zgłoszeń, np. diagnostyki błędu i przypisania osoby odpowiedniej

do rozwiązania zgłoszenia, identyfikowania duplikatów zgłoszeń itp. W większości projektów nowe zgłoszenie lub błąd jest ręcznie oceniane przez doświadczonego programistę lub analityka. Wstępna analiza może być wysoce czasochłonna z uwagi na niepoprawne przypisanie zgłoszenia do niewłaściwej osoby i potrzebę przeniesienia go kolejnego operatora (programisty, analityka). Czas ten może zostać zminimalizowany przy zastosowaniu automatycznego przypisywania zgłoszeń do rekomendowanych programistów na podstawie klasyfikacji tekstu opisanego w [47]. Podejście to opiera się na łączeniu zgłoszenia błędu z programistą na podstawie szeregu wyznaczanych atrybutów. Inne rozwiązania dyskutowane są w [45]. Autorzy sprawdzają czy nowo dodane zgłoszenie jest duplikatem oraz jeżeli tak, wskazują 20 możliwych duplikatów. W przeciwnym wypadku rekomendowany jest doświadczony programisty do którego mogłoby zostać przydzielone weryfikowane zadanie. Przypisywanie zgłoszenia w połączeniu z analizą istotności zgłoszenia dyskutowane jest w [92].

Przegląd technik automatycznego zarządzania zgłoszeniami błędów poprzez: detekcję duplikatów, określanie priorytetu, przypisywanie zgłoszeń przedstawiony został w [27]. Badanie rozszerzone zostało o możliwość oceny poprawności zgłoszenia błędu z wykorzystaniem algorytmu lasu losowego. Ważną kwestią możliwych usprawnień procesu obsługi błędów jest automatyczne wykrywanie duplikatów zgłoszeń w celu wyeliminowania nadmiarowych aktywności interesariuszy oraz minimalizacji czasu spędzanego na przypisywaniu zgłoszeń. Liczba duplikatów w zależności od projektu może stanowić od kilku nawet do 30% wszystkich zgłoszeń. W moim doświadczeniu praktycznym z projektami komercyjnymi zaobserwowałem problem pomijania raportowania wykrytych duplikatów we wczesnym etapie analizy zgłoszenia. W [93] autorzy sprawdzają podobieństwo nowo tworzonego błędu na podstawie podobieństwa semantycznego i leksykalnego z już istniejącymi zgłoszeniami w celu wyliczenia prawdopodobieństwa duplikacji. Praca [94] przedstawia pewne metody określania duplikatów, bazując na różnych metodach: TF-IDF, uczenie maszynowe, analiza tytułu oraz głębokiego uczenia. W [45] autorzy badają podobieństwo opisu zgłoszenia oraz tytułu według cosinusowej metryki podobieństwa. Większość z prac poświęconych wykrywaniu duplikatów bazuje na technikach przetwarzania języka naturalnego. Jedynie kilka prac [95] bierze pod uwagę także informacje dotyczące logów błędu z wykorzystaniem śladu stosu (ang. *stack trace*).

Zaprezentowany przegląd literatury potwierdza praktyczne znaczenie technik eksploracji tekstu we wspieraniu procesu obsługi błędów w cyklu życia projektu.

Omówione badania ukierunkowane są na algorytmy dostosowane do specyficznych problemów i pozbawione są szczegółowych badań semantycznych i strukturalnych cech raportowanych zgłoszeń oraz ich komentarzy. Analizując wiele repozytoriów projektowych, zaobserwowałem, że zawierają one wysoki procent słów nie będących częścią języka naturalnego. Wprowadzając taksonomię tych fraz, opracowałem szereg wyrażeń regularnych pozwalających na ich grupowanie. Zastąpienie znalezionych fraz etykietami odpowiednich kategorii zwiększa ich wpływ na semantykę tekstów oraz wprowadza nowy wymiar do klasyfikacji tekstów i ich interpretacji. Prezentuję szereg nowych cech z eksploracji tekstu, które możemy połączyć z cechami statystycznymi i obsługi procesu zgłoszeń, co również jest pomijane w dostępnej literaturze. Cechy te mogą być interpretowane w szerszym kontekście zależności obsługi zgłoszeń, dlatego zaprezentowana analiza jest bardziej ukierunkowana na ocenę projektu. W rozdziale omówiłem szerokie studium tych przypadków, które zwiększa przestrzeń oceny obsługi zgłoszeń i ich dokumentacji.

4.2. Analiza słowników

Po przeanalizowaniu wielu repozytoriów zgłoszeń (ITS) projektów, zarówno *open source* jak i komercyjnych, zaobserwowałem, że słowa w nich występujące to mieszanka słów języka naturalnego (należącego do tezaurusa) oraz innych słów niebędących częścią tezaurusa, w których mogą znajdować się różne grupy słów, jak np. żargon projektowy, nazwy funkcji bądź odniesienia do kodu aplikacji, link do załącznika, link do zewnętrznego zasobu, część kodu, logi aplikacji, znaczniki formatowania tekstu w repozytorium itp. Dodatkowo tekst może być obarczony błędami w pisowni wynikającymi z literówek bądź używania innego języka. Wszystkie wymienione typy słów wpływają na jakość tekstu. Przedstawiają też ciekawą informację, która została ujęta w przeprowadzonych przeze mnie analizach. W ramach prac opracowałem szereg słowników pozwalających na grupowanie poszczególnych słów z opisów, tytułów oraz komentarzy zgłoszeń. Wyróżniłem 4 słowniki: słownik słów należących do tezaurusa (NLW), słownik słów projektowych (FW), słownik słów niesklasyfikowanych (NCW), słownik słów zgodnych z notacją *CamelCase* (CCW). Słownik NLW w większości przypadków będzie zawierał elementy występujące w słowniku tezaurus. Jednak w analizowanym tekście mogą znaleźć się słowa innych języków, np. słowa polskie, z których można zbudować dedykowany słownik lub, jeżeli są sporadyczne, mogą zostać uznane za część słownika NCW. Mogą znaleźć się w nim także słowa, których znaczenie nie zostało określone bądź są błędami w pisowni słów ze słownika tezaurus. Słownik FW składa się z określeń dedykowanych dla danego projektu bądź etykiet wyrażeń specjalnych,

takich jak odnośniki, nazwy klas, logi aplikacji, znaczniki formatowania tekstu w JIRA, odnośniki do zmian w kodzie (*PR*).

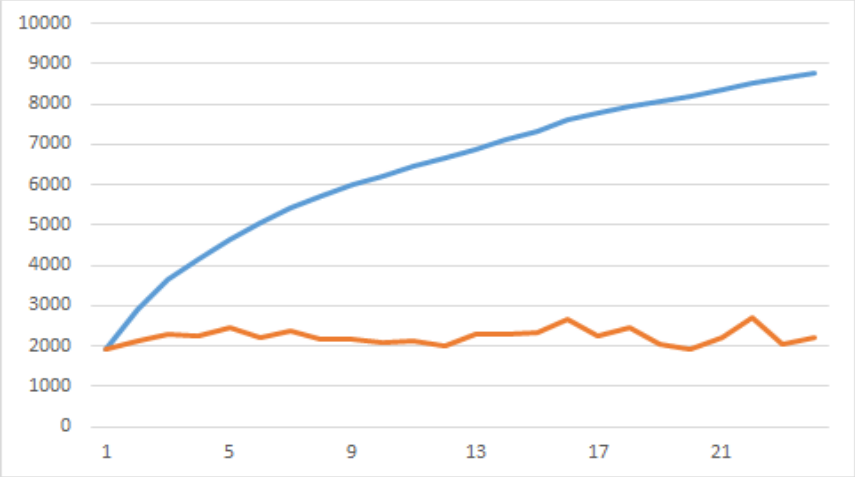
Wyrażenie regularne	**Przeznaczenie**
\[~\S+\]	Nazwa użytkownika JIRA
\!\S+[.png|.jpg|.gif]\!	Załącznik graficzny
\!\S+.txt\!	Załącznik tekstowy
\!\S+.log\!	Załącznik pliku log
\!\S+\!	Załącznik z dowolnym rozszerzeniem
(\[(.)*?pull-request?.*\])| (https:\/\/.*?pull-request[\S]*)| (\[(.)*?\/pull\/?.*\])| (https:\/\/.*?\/pull\/[\S]*.?)| (https:\/\/.*?\/commit\/[\S]*.?)	Odnośnik do pull request (PR)
(\[\S+|\S+\])|(\[http.*?\])| (\[https:\/\/\S+)|(https:\/\/\S+)| (http:\/\/\S+)	Odnośnik zewnętrzny
{panel(.|\n)*?{panel}	Sekcja panel
({code(.|\n)*?{code})| ({noformat}(.|\n)*?{noformat})| ({code:java}(.|\n)*?{code})	Sekcja kodu
(([a-zA-Z_$][a-zA-Z\d_$]*\.){1,}[a-zA-Z_][a-zA-Z\d_$]*)| (([a-z]|[A-Z])+(_([a-z]|[A-Z])*)+.js)| (\b([a-z]|[A-Z])+(_([a-z]|[A-Z])*)+\b)	Nazwa klasy lub pakietu kodu (włączając pliki js)
(?:[a-z0-9!#$%&*+/=?^_`{\|}~-]+(?:\.[a-z0- 9!#$%&*+/=?^_`{|}~-]+)*|"(?:[\x01-\x08\x0b\x0c\x0e- \x1f\x21\x23-\x5b\x5d-\x7f]|\\[\x01-\x09\x0b\x0c\x0e- \x7f])*")@(?:(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](?:[a- z0-9-]*[a-z0-9])?|\[(?:(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1- 9]?[0-9]))\.){3}(?:(2(5[0-5]|[0-4][0-9])|1[0-9][0-9]|[1-9]?[0- 9])|[a-z0-9-]*[a-z0-9]:(?:[\x01-\x08\x0b\x0c\x0e-\x1f\x21- \x5a\x53-\x7f]|\\[\x01-\x09\x0b\x0c\x0e-\x7f])+)\])	Adres E-mail. Komentarz: wyrażenie regularne zapożyczone zostało z https://stackoverflow.com/a/201378
\w*[A-Z]\w*[A-Z]\w* \b[a-z]+[A-Z]+\S*\b	Nazwa klas bądź metod skonstruowanych zgodnie z koncepcją CamelCase (duże litery na początku nowego członu nazwy klasy/metody)
([.,*#]\s)|([,:;!%&<=>{}|~])	Znaki interpunkcyjne

Tab. 11 Przykłady wyrażeń regularnych definiujące obiekty tekstowe spoza klasy NLW.

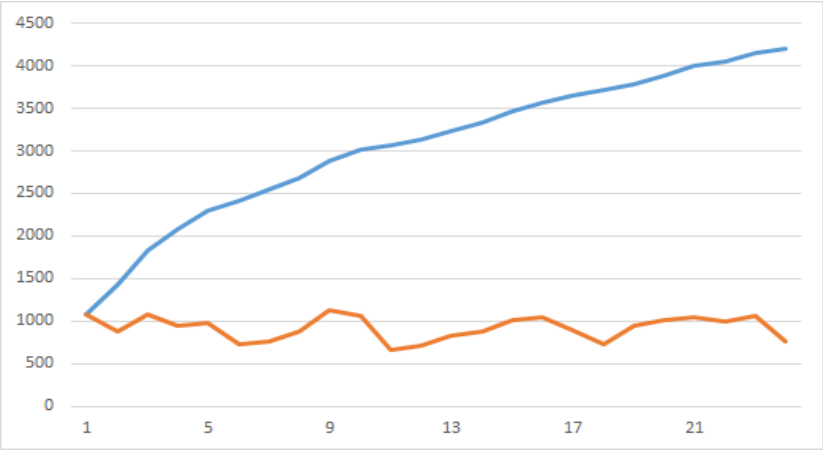
Identyfikacja składowych słownika FW (nazwy własne, frazy, wycinki kod, itp.) dokonywana jest na podstawie wyrażeń regularnych (Tab. 11) wykorzystujących notację silnika JavaScript [96]. Opracowałem je na podstawie obserwacji tekstów dostępnych w repozytoriach. Słownik CCW składa się z: nazwy klas, modułów, zmiennych lub metod kodu, które zapisywane są za pomocą sformalizowanej notacji, np. *CamelCase, snakecase,*

PascalCase. W większości sprawdzonych projektów zaobserwowałem notację *CamelCase*, dlatego tokeny te klasyfikowane są do dedykowanego słownika CCW. Różnią się one od nazw klas bądź pakietów zapisywanych, np. *java.lang.Object, java.lang.Number*, które wykrywam za pomocą wyrażeń regularnych i zliczam w słowniku FW. Pseudokod algorytmu budowania słowników przedstawiłem w Alg. 1. Wszystkie zaprezentowane w pracy algorytmy przedstawione są w postaci pseudokodu opartego o uproszczoną notację języka *Python*. Wyrażenia regularne wspierają generowanie struktury słowników, które dają ogólny pogląd na złożoność dalszych algorytmów eksploracji tekstu oraz jakość raportów zgłoszeń.

Istotną perspektywą w analizie słownikowej jest weryfikacja ich objętości. Szczególnie wartościowe wydaje się spojrzenie krótko i długookresowe. Wykresy (Rys. 2, Rys. 3) przedstawiają przykład perspektywy przyrostowej i okresowej objętości słownika NLW (tezaurus) dla zgłoszeń błędów projektu MongoDB (Rys. 2) oraz P1 (Rys. 3) w okresie 24 miesięcy, w których dodano odpowiednio 5469, 2865 raportów błędów.



Rys. 2 Objętość przyrostowa i okresowa słownika NLW (tezaurus) dla zgłoszeń błędów projektu MongoDB



Rys. 3 Objętość przyrostowa i okresowa słownika NLW (tezaurus) dla zgłoszeń błędów projektu P1

Górna linia przedstawia wartość skumulowanego słownika, dolna – perspektywę poszczególnych miesięcy. Interesująca jest relatywna stabilność wielkości unikalnego słownika NLW dla projektu MongoDB w ujęciu miesięcznym (1916-2648, średnio 2232). Projekt P1 wykazuje analogiczną tendencję, jednak różnica wielkości słowników w poszczególnych miesiącach jest większa; zakres słownika NLW wynosi 656-1127, średnio 921. Wielkość słownika P1 w ujęciu skumulowanym (po 24 miesiącach z wartością 4205) jest uboższa od słownika NLW projektu MongoDB (po 24 miesiącach osiągnął wartość 8779), jednak w obydwu przypadkach następuje jego stabilizacja w dłuższej perspektywie czasowej. Różnica między wielkością słownika wynika z wielkości projektu oraz zakresu ich funkcjonalności. Wpływ na słowniki NLW może mieć również liczba aktorów zaangażowanych w proces oraz ich aktywność. W projekcie P1 zauważalna jest mała fluktuacja zespołu, co prowadzi do wypracowania wspólnego języka – żargonu projektowego specyficznego dla branży. Projekt MongoDB posiada szeroką listę reporterów (około 1200 na przestrzeni 3 lat), wskutek czego ma tendencję do używania różnych słów słownika NL. Weryfikacja ręczna słowników w szczególności FW, NCW byłaby żmudna i czasochłonna, dlatego opracowałem Alg. 1 wspierający ten proces z wykorzystaniem słownika tezaursua (155287 słów i 117659 synonimów dla języka angielskiego), oraz algorytmu pomocniczego (Alg. 2) przeprowadzającego sprawdzenie pisowni (metoda *checkSpellingError*) oraz tagowanie części mowy (POS, ang. *part of speech tagging metoda tag()*) poszczególnych wyrazów. Alg. 2, zaimplementowany został z wykorzystaniem biblioteki *NLTK* [97]. Profile słownikowe mogą być wygenerowane dla pewnych ograniczeń (R) np. grup aktorów (reporterów), typów zgłoszeń, priorytetu zgłoszeń itp.

```
input: text of issue description or comment content as text parameter
output: tuple with values: list of processed words, dictionaries of NLW, FW,
CCW, NCW for given text
```

```
1 function generate_dictionaries(text):
2    list = new empty list
3    NLW, FW, CCW, NCW, tagW = new empty dictionary
4    txtT = replaceRegExpwithTags(text)
5    [T] = tokenize_word(txtT)
6    FOR t in [T] DO
7        IF t is not digit or len(t) > 2 THEN
8            IF t.toLowerCase() not in stop_words THEN
9                tmpList.add(t)
10           ENDIF
11       ENDIF
12    FOR t in tmpList DO
13       IF t in tezaurus THEN
14           IF t in NLW THEN
15               NLW[t] += 1
16           ELSE:
17               NLW[t]=1
18           ENDIF
19       ELSEIF t in tagList:
20           IF t in tagW THEN
21               tagW [t] += 1
22           ELSE:
23               tagW [t]=1
24           ENDIF
25       ELSEIF t match CamelCaseRegExp:
26           IF t in CCW THEN
27               CCW[t] += 1
28           ELSE:
29               CCW[t]=1
30           ENDIF
31       ELSEIF is_valid_tag(token)
32           IF t in FW THEN
33               FW[t] += 1
34           ELSE:
35               FW[t]=1
36           ENDIF
37       ELSE:
38           IF t in NCW THEN
39               NCW[t] += 1
40           ELSE:
41               NCW[t]=1
42           ENDIF
43       ENDIF
44    ENDFOR
45    resultTuple = new tuple(list, NLW, FW, CCW, NCW)
46    return resultTuple
47 endfunction
```

Alg. 1 Pseudokod generowania słowników *NLW*, *FW*, *CCW*, *NCW*

Jako parametr wejściowy Alg. 1 przyjmuje tekst, z którego budowane są opracowane słowniki. Tekst wejściowy przetwarzany jest na pojedyncze tokeny (*linia 5*), z których usuwane są frazy składające się wyłącznie z cyfr krótszych niż 2 znaki bądź będące słowem przystankowym (ang. stop words) (*linie 6-11*). Przefiltrowana lista tokenów jest sprawdzana

kolejno względem: słownika tezaurus (*linia 13*), listy specjalnych tagów budowanych na podstawie wyrażeń regularnych (*linie 19-24*), wyrażenia regularnego sprawdzającego notację CamelCase (*linia 25*), poprawność tagu (*linia 31*, pseudokod metody *is_valid_tag* przedstawia Alg. 2). Sprawdzane warunki determinują przynależność tokenu do opracowanych słowników odpowiednio dla NLW, CCW, FW, NCW, które zwracane są wraz z przetworzonym tekstem na wyjściu Alg. 1 (*linia 45*).

Opracowane profile zakładają analizę w ujęciu czasowym, dla których Alg. 1 jest wykonywany dla każdego opisu zgłoszenia, a jego wynik jest agregowany. Różnorodność słowników NCW i FW projektów P1 oraz MongoDB w ujęciu miesięcznym jest o wiele mniejsza niż NLW, jednak statystyka skumulowana w perspektywie długookresowej, wykazuje ich liniowy przyrost. Badając takie zbiory, wartościowe jest odniesienie do zgłoszeń błędów oraz długości opisów poszczególnych raportów zgłoszeń.

```
input: token (T) (word tokenized from text)
output: Boolean value defining if algorith has recognized word as project
specific word
```

```
1 function is_valid_tag(T):
2    TAG = tagger.tag(T)
3    IF (checkSellingErorr(T)='None' and TAG ='O') OR TAG == 'ORGANIZATION'
        OR TAG == 'PERSON' THEN
4        return TRUE
5    ELSE
6        return FALSE
7    ENDIF
8 endfunction
```

Alg. 2 Pseudokod funkcji pomocniczej is_valid_tag

W okresie 24 miesięcy dla projektu P1 zgłaszano miesięcznie 73-170 (średnio 119) błędów, w których liczba słów wynosiła 3984-9225 (średnio 6465). Oznacza to, że jeden opis raportu zgłoszenia błędu składał się z 54 tokenów. Algorytm pozwala także na wykrycie słów niesklasyfikowanych, które w projekcie P1 stanowiły średnio 4,66%. Możemy oszacować średnią liczbę takich słów w opisie zgłoszeń błędów projektu P1 jako 2. Daje nam to pewien pogląd na jakość dodanych opisów.

Profil jakości opisów (*PJO*) może specyfikować przez wektor :

$$PJO = \langle \text{średnia liczba zgłoszeń w miesiącu; średnia liczba tokenów w zgłoszeniach; procent niesklasyfikowanych tokenów} \rangle \quad (4.1)$$

Który dla projektu komercyjnego P1 oraz projektu *open source* MongoDB wynosi odpowiednio:

$PJO_{P1} = \langle 119; 6465; 4{,}66\% \rangle$

$PJO_{MongoDB}$ = <228;21385;9,45%>

Profil może być wyznaczony w perspektywie długookresowej oraz krótkookresowej w celu monitorowania jakości opisów. Poziom fluktuacji niesklasyfikowanych tokenów w projekcie MongDB wahał się w zakresie 8,17% - 11,7%. Istotne jest również sprawdzenie liczności słownika słów niesklasyfikowanych. W projekcie P1 średnia wartość miesięczna wynosiła 70, z fluktuacją w zakresie 46-94. Liczba niesklasyfikowanych tokenów z 24 miesięcy wynosiła 685 unikalnych słów. Warto, aby zbiór ten był weryfikowany ręcznie w celu poszukiwania możliwych nowych klas słów, na podstawie których można opracować nowe wyrażenia regularne, poprawiające ostateczną klasyfikację. Sumaryczna wielkość słownika niesklasyfikowanych tokenów w projekcie MongoDB dla 24 miesięcy wyniosła 1780, ze średnim miesięcznym przyrostem na poziomie 238. Statystyka ta może być wyznaczona dla różnych typów zgłoszeń.

Raporty zgłoszeń projektowych zawierają w sobie specyficzne teksty, które mogą być częścią słownika FW budowanego za pomocą wyrażeń regularnych Tab. 11. Spośród specjalnych fraz wyróżniam mi.in. adresy email, wycinki kodu, załączniki binarne, załączniki obrazów, linki, odnośniki do zmian w kodzie (ang. *pull request*) oraz panele JIRA. Przykłady ich wystąpień w rozpatrywanych projektach P1, MongodDB w czasie 24 miesięcy ujęte zostały w Tab. 12.

Projekt		Email	CS	CN	BA	IA	L	PR	JP
MongoDB	Suma	38 (1%)	1276 (28%)	2276 (51%)	247 (5%)	91 (2%)	393 (9%)	114 (3%)	61 (1%)
	Fluktacja	0-7	13-117	22-249	0-33	0-29	5-48	0-11	0-12
P1	Suma	2359 (51%)	30 (1%)	669 (30%)	1012 (64%)		561 (91%)		
	Fluktacja	43-173	0-10	0-199	10-98		6-40		

Tab. 12 Statystyka słów specjalnych projektów MongoDB i P1

Oznaczenia kolumn w tabeli: E – email, CS – wycinek kodu, CN – nazwa klasy/pakietu, BA – załącznik binarny, IA – załącznik obrazkowy, L – link, PR – odnośnik do zmiany w kodzie, JP – JIRA panel. W nawiasach podaję procent tagów względem ich całkowitej liczby

Statystyka zaprezentowana w Tab. 12 daje istotny wgląd w użycie klas znaczników specjalnych w badanym okresie. Dane te są istotne w dalej zaprezentowanych rozważaniach dotyczących opracowanych algorytmów klasyfikacji. Możemy także ocenić na ich podstawie poziom jakości opisów zgłoszeń. Duża liczba nazw klas (*CN*) świadczy o bardziej technicznym

opisie zgłoszenia. Projekt MongoDB ma większy stopień udziału wyrażeń określających nazwy klas (CN) niż projekt komercyjny P1 odpowiednio 51% i 1%. Porjekt P1 natomiast ma o wiele większy udział załączników binarnych (64%) oraz linków (91%) w opisach zgłoszeń błędów. Różnica wynika ze specyfiki projektów. W zgłoszeniu błędu projektu mongoDB pojawiają się nazwy klas z którymi związane są techniczne błędy, w projekcie P1 natomiast często występują załączniki nagrań kroków prowadzących do błędu w tworzonej aplikacji lub linków do środowisk na których występuje błąd. Potwierdza to wartość informacji zawartych w analizie słowników słów specjalnych opisów zgłoszeń, co wykorzystałem w opracowanym mechanizmie klasyfikacji który przedstawiłem w sekcji 4.3.

Profil jakości opisów może zostać rozszerzony o informację o liczbie wystąpień słów słownika FW. Definiujemy w ten sposób specyficzny dla projektu opis zgłoszeń. Profil może zostać zapisany w postaci wektora *<liczba zgłoszeń, wielkość słownika NL, wielkość słownika słów niesklasyfikowanych, liczba wystąpień specjalnych klas słów>*. Opracowane profile mogą zostać wyznaczone dla zgłoszeń z różnymi ograniczeniami np. typu zgłoszenia, grupy reporterów itp. Dla przykładu wykorzystałem projekt P1, w którym profil dla zgłoszeń typu: *User Story, New Feature, Task, Bug* w okresie 6 miesięcy reprezentowane są przez odpowiednie wektory *{138, 1429, 80, 635}, {128, 1976, 81, 473}, {195, 1139, 81, 695}, {244, 1206, 117, 953}*. Wektory dają ogólny pogląd na specyfikę występujących opisów w różnych typach zgłoszeń. Możemy zgłębić się w poszczególne wartości w perspektywie zgłoszenia i oszacować wystąpienia poszczególnych klas słów specjalnych, których średnia wartość wystąpień nazwy klas w rozpatrywanych typach zgłoszeń wynosiła odpowiednio 1,36; 1,91; 1,67; 0,40. Dla zgłoszeń nowych funkcjonalności (*User Store, New Feature, Task*) współczynnik ten jest o wiele wyższy niż dla zgłoszeń błędów, co wskazuje skonkretyzowany charakter zgłoszeń.

Opisy zgłoszeń mogą składać się także ze słów technicznych (TW), adekwatnych do dziedziny projektu. Zasadne jest ich zidentyfikowanie i zinterpretowanie. Część z nich może zawierać słowa ze słownika NLW bądź innych klas słów. Za pomocą słów TW możemy skorelować analizowany tekst z konkretnym tematem, np. błędów wydajnościowych bądź wykonania określonej funkcjonalności. Słowa TW mogą być także rozszerzone do n-gramów odgrywających podobną rolę jak słowa kluczowe. Jest to użyteczne w wyznaczaniu tytułu zgłoszenia. Dla ilustracji podałem przykłady takich słowników w języku angielskim:

- *Problemy wydajnościowe*: enhance, add, ignore, improve, optimize, performance, required, support, can, may, suggest, slow, stuck, loading time, time-consuming.

- *Problemy bezpieczeństwa*: access, admin, grant, privileges, roles, revoke, security, firewall, blocked, security issue, ssl, certificate, authorization.
- *Problemy pamięci*: cache, crash, heap, infinite, segmentation, segfault, heap dump, memory leak.
- *Błędy logiczne*: assertion, argument, break, error, exception, broken, incorrect, incomplete, pointer, parameter, remove, statement, status, wrong, not proper, fail, null, null pointer, use case, logic flow.

Zarówno zbiory słów, jak i odpowiednie n-gramy, mogą być wyprowadzane iteracyjnie i aktualizowane. Mogą być także tematem wewnętrznych dyskusji zespołu projektowego w celu ustandaryzowania i poprawiania opisów, włączając listę słów kluczowych. Mogą one zostać także skorelowane z algorytmami klasyfikacji opartymi o uczenie maszynowe. Omówione słowniki stanowią część wyznaczonych cech użytych w algorytmach klasyfikacji przedstawionych w kolejnym rozdziale.

W opisach zgłoszeń występują złożone sekcje, które mogą być wyszczególnione za pomocą wyrażeń regularnych, np. wycinki kodu. Niemniej, pojawiają się także mniej sformalizowane sekcje (słabo ustrukturyzowane). Zazwyczaj dotyczą one opisu problemu z wyszczególnieniem rzeczywistego i oczekiwanego rezultatu lub kroków niezbędnych do odtworzenia problemu. Przedstawione statystyki tekstowe są użyteczne w ocenie wartości informacyjnej oraz ich strukturze. Mogą także wskazać potencjalne miejsca do poprawy raportowania.

4.3. Schematy klasyfikacji

Jednym z najczęściej występujących problemów w czasie prowadzenia projektów informatycznych jest poprawna klasyfikacja zadań dodawanych do repozytorium zgłoszeń. Każdego dnia może powstawać nawet kilkaset nowych zgłoszeń, a każde z nich powinno zostać zweryfikowane przez analityka bądź inną osobę odpowiedzialną za wstępną walidację zgłoszenia. Częstym problemem klasyfikacji jest niepoprawne określenie typu zgłoszenia przez osobę dodającą zgłoszenie do repozytorium. Zgłoszenie błędu może być nową funkcjonalnością bądź innym typem zgłoszenia, np. technicznym zadaniem dotyczącym wprowadzenia konfiguracji (“*TASK*”). Managerowie delegują zadanie wstępnej analizy zgłoszeń pomiędzy różnych aktorów projektowych, bądź zatrudniają do tego celu dedykowaną osobę. Aktywność ta związana jest z wysokim zaangażowaniem wybranego aktora, co znacząco wpływa na inne zadania, do których mogłaby zostać oddelegowana dana osoba. Czasami ręczna klasyfikacja zgłoszenia jest błędna. Rozwiązaniem tych problemów może być

automatyczna klasyfikacja zgłoszeń. Biorąc pod uwagę wyniki analizy przedstawione w sekcji 4.1. postanowiłem opracować nowe modele klasyfikacji na podstawie wstępnie przetworzonego tekstu, zgodnie z klasycznymi metodami eksploracji tekstów, wzmocnionymi autorskimi transformacjami opisów tekstowych. Przekształcony tekst jest poddawany schematom klasyfikacji dostosowanym do celów analizy semantycznej (z sekcji 4.3.1). Niektóre przykładowe wyniki zaproponowanego oryginalnego podejścia zostały przedstawione w sekcji 4.3.2.

4.3.1. Algorytmy

Opracowany schemat eksploracji zawartości tekstowej repozytoriów programowych wykonywany jest w dwóch fazach: 1) wstępne przetwarzanie tekstu, 2) klasyfikacja. Kroki wstępnego przetwarzania można przedstawić poprzez ciąg operacji:

1. Pobranie zgłoszeń z repozytorium poprzez API JIRA (mogą dotyczyć zdefiniowanego zakresu czasu)
2. Stworzenie zbioru oryginalnych encji tekstów oznaczonych identyfikatorem zgłoszenia lub komentarza
3. Redukcja tekstu poprzez usunięcie słów przystankowych oraz cyfr niepołączonych z innymi znakami.
4. Transformacja oryginalnych encji tekstów z wykorzystaniem wyrażeń regularnych oraz wyznaczenie opracowanych cech tekstowych. Możemy wyróżnić dwie opcje tego kroku, pierwsza zastępująca wykryte frazy wyrażeń regularnych ogólnymi tagami (np. *code, user, email*) słownika FW lub druga opcja zastąpienie fraz sparametryzowanymi tagami określającymi ich unikalność np. *user#1, user#2*.

Pseudokod opisanych kroków z wyłączeniem pobrania zgłoszeń przedstawia Alg. 3.

```
input: List of issues ([I]) to be classified with list of features ([F])
which needs to be calculated. For title it is TL - title lenght, TS - title
senthiment, for description: DL - description legth, DS - description senthi-
ment, %TW - percent of technical words,|PR| - count PR tags, |L| - count of
links tag etc.
output: Matrix (DataFrame) with preprocessed text and calculated features for
all issues from input
```

```
1 function process_issues([I], [F]):
2    result = new empty Matrix (python DataFrame object of Pandas library)
3    FOR i in [I] DO
4        T = preproces(i.title)
5            FOR regExp in listOfRegExps DO
6                T.replace(regExp.pattern, regExp.tag)
7            ENDFOR
8        D = preproces(i.description)
9            FOR each regExp in listOfRegExps DO
10                D.replace(regExp.pattern, regExp.tag)
11           ENDFOR
12       [TF] = calculateFeatures(title, featureList.title)
13       [DF] = calculateFeatures(description, featureList.description)
14       [CF] = merge [TF] and [DF] into one row
15       result.append([CF])
16    ENDFOR
17    return result
18 endfunction
```

Alg. 3 Pseudokod przetwarzania tekstu na potrzeby klasyfikacji

W przeciwieństwie do klasycznych technik eksploracji tekstu w Alg. 3 nie zastosowałem unifikacji procesowego tekstu do małych bądź dużych liter, z uwagi na istotność formatu liter w wykrywaniu nazw zmiennych, klas, modułów lub pakietów kodu w notacji *CamelCase*. Funkcja *preproces (linie 4,8)* usuwa z tekstu tokeny krótsze niż 2 znaki oraz cyfry jeżeli nie są połączone z literami. Funkcja *calculateFeatures* (*linie 12,13,14*) reprezentuje mechanizm obliczania autorskich cech takich jak: długość opisu zgłoszenia (z zastosowaniem metody *T-shirt sizeing* – skomentowane w dalszej części), długość tytułu zgłoszenia (metoda T-shirt size), sentyment opisu (przyjmuje trzy wartości: Pozytywny, Negatywny, Neutralny), sentyment tytułu zgłoszenia, Liczba etykiet zgłoszenia (Liczba etykiet w polu *Label*), Liczba wersji wpływu (Liczba wersji w polu *Affected version's* zgłoszenia), Liczba referencji do zmian w kodzie, Liczba odnośników zewnętrznych, Liczba nazw klas kodu, Liczba wycinków kodu, Liczba paneli JIRA w opisie, Liczba załączonych logów, Liczba załączników tekstowych, Liczba załączników binarnych, procentowy udział słów słownika TW, liczba adresów e-mail, liczba wystąpień słów słownika FW. Łącznie opracowałem 18 dodatkowych cech tekstów, które wyznaczane są na podstawie treści zgłoszenia oraz właściwości tekstów. Algorytm daje również możliwość wyznaczenia tych cechy, dla treści komentarzy zgłoszeń.

W zbiorze opracowanych cech znajdują się zarówno pola pobrane bezpośrednio z definicji zgłoszenia JIRA, jak i cechy wyznaczone na podstawie opisu i tytułu zgłoszenia przez autorski mechanizm Alg. 3**.** Do ekstrakcji danych wykorzystane zostały wyrażenia regularne (Tab. 11), metryka sentymentu określająca charakterystykę tekstu (przy wykorzystaniu funkcjonalność Pakietu *NLTK[97] SentimentIntensityAnalyzer*). Mechanizm wykorzystuje słownik zwrotów negatywnych oraz pozytywnych wraz ze zwiększeniem znaczenia specjalnych fraz zdefiniowanych w bibliotece tzw. *boosting*. Bazując na wystąpieniach poszczególnych grup słów, algorytm wyznacza wskaźnik sentymentu w zakresie od -1 do 1. Wartości z tego zakresu mapowane są do klas: Pozytywny, Negatywny, Neutralny. Podobnie działa algorytm podziału długości tytułu, bądź opisu z podziałem na 3 grupy względem zdefiniowanych zakresów. Określenie długości tekstów oparłem o tzw. metodę „*t-shirt sizing*", zaczerpniętą z metodologii *SCRUM* [98]. *T-shirt sizing* jest jedną z dostępnych metod estymacji, w której zespół zamiast określania konkretnych punktów historyjek (ang. *story points*), określa jedynie rozmiar zadania i poziom jego skomplikowania poprzez rozmiary koszulek *XS, S, M, L, XL*.

Wyekstrahowane teksty raportów zgłoszeń były weryfikowane przeze mnie ręcznie w celu zbudowania zbioru trenującego opracowywanej klasyfikacji. Starałem się, aby zbiór trenujący był jak najbardziej zbalansowaną reprezentacją wszystkich klas docelowych.

Alg. 4 przedstawia pseudokod opracowanego mechanizmu wyboru najlepszego klasyfikatora, który na wejście przyjmuje listę algorytmów do walidacji ([*A*]), zbiór trenujący (*trainingSet*) oraz listę list cech ([*F*]), które mają zostać zweryfikowane. W publikacji [40] te cechy nazwałem konfiguracją danych. Wynikiem działania algorytmu jest finalny klasyfikator (algorytm i zestaw cech) z najlepszym wynikiem precyzji oraz macierzą z wynikami wszystkich przeprowadzonych testów względem algorytmów i listy cech. Przed przystąpieniem do klasyfikacji mechanizm przekształca cechy na wektory liczbowe przy zastosowaniu odpowiednich transformatorów (*linie 7-10*). Cechy tekstowe interpretowane są poprzez implementację mechanizmu TF-IDF dostępnego w bibliotece *sklearn* [99] (*TfidfVectorizer*).

```
input: list of algorithms [A], training set [S], list of features set [F]
output: Tuple with best classifier (FC) and matrix of all classification
results (RM) where row represents tested features and columns relates to ap-
plied algorithm
```

```
1 function choose_classifier([A], trainingSet, [F]):
2    resultList = new Empty list
3    RM = empty matrix //Python DataFrame object from Pandas//
4    TRS, TS = trainingSet.split(0.7)
5    FOR F IN [F] DO
6        FR = new empty SET
7        CT = define a column transformer for F
8                          //tfidfVectorizer -  for text features//
9                          //OneHotEncoder - for labels features//
10                         //StandardScaler - for numerical features//
11       FOR A IN [A] DO
12             C = new Classifier(A, F)
13             C.crossValidation(A*, TRS) //with GridSearchCV:
14             C.train(TRS)
15             SC = C.validate(TS)
16             FR[A]= <SC, C>
17       ENDFOR
18       RM[F] = FR
19   ENDFOR
20   FC = RM.getBestACC()
21   return Tuple(FC,RM)
22 endfunction
```

Alg. 4 Pseudokod klasyfikacji

Metryka TF-IDF używana jest do wyliczania macierzy podobieństw między zweryfikowanymi tekstami. Algorytm wyliczania TF-IDF składa się z 3 kroków: I – wyliczenie TF, II – obliczenie IDF, III – obliczenia TF-IDF według wzorów [100]:

$$tf - idf = tf(t,d) * idf(t,D) \quad (4.2)$$

Gdzie $tf(t,d)$ oznacza częstość wystąpień termu t w dokumencie d:

$$tf(t,d) = \frac{f(t,d)}{|d|} \quad (4.3)$$

Gdzie: $f(t,d)$ liczba wystąpień termu t w dokumencie d, a $|d|$ to liczba wszystkich termów w dokumencie d

$$idf(t,D) = log\frac{|D|}{|\{d|t \epsilon d\}|} \quad (4.4)$$

Gdzie mianownik określa liczbę wszystkich dokumentów, w których występuje term t.

Wartość metryki TF określa częstość wystąpień wyrazu w dokumencie. Wartość ta obliczana jest jako stosunek liczby wystąpień wyrazów do całkowitej liczby wyrazów

w dokumencie. Współczynnik IDF oznacza odwróconą częstość wystąpień wyrazów w dokumentach. Używany jest do obliczania wystąpień rzadkich wyrazów pośród wszystkich badanych dokumentów. Wyższy współczynnik IDF oznacza rzadsze występowanie słowa w dokumencie. Współczynnik TF-IDF umożliwia wyznaczenie różnic pomiędzy poszczególnymi dokumentami. Metoda TF-IDF znalazła swoje szerokie zastosowanie w *text minungu* [101], co potwierdza jej przydatność.

Do przetwarzania cech etykietowych (kategorii) wykorzystany został mechanizm *OneHotEncoder* z pakietu *sklearn[99]*, który wykonuje kodowanie „1 z n". W uproszczeniu, algorytm wyznacza macierz binarną, w której każdej etykiecie (kategorii) odpowiada ciąg bitów, w którym występuje tylko jedna 1, a pozostałe bity mają wartość 0.

Cechy numeryczne podlegają standaryzacji przy użyciu implementacji *StandardScaler* pakietu *sklearn[99]*. Zgodnie z dokumentacją, znormalizowana wartość cechy wyliczana jest na podstawie wzoru $z = (x - u) / s$, gdzie x oznacza wartość próbki, u średnią wartość zbioru uczącego, zaś s określa standardowe odchylenie zbioru uczącego cechy.

Możliwe jest połączenie wielu cech klasyfikacji bądź przeprowadzenie weryfikacji wszystkich możliwych permutacji cech. Jednak wydaje się to nieoptymalne z uwagi na liczbę niezbędnych obliczeń. Dlatego istotne jest, aby osoba znająca dobrze projekt dokonała wstępnej preselekcji grup które będą użyte przy wyznaczaniu najlepszego klasyfikatora przez opracowany algorytm (Alg. 4). Wstępna ocena użyteczności cech może zostać dokonana na podstawie analiz statystycznych przedstawionych w sekcji 4.2. Bazując na doświadczeniu z przeprowadzonych eksperymentów, zweryfikowałem maksymalnie do 4 dodatkowych cech wraz z treścią opisu i tytułu zgłoszenia.

Proces trenowania klasyfikatora oparty jest o technikę walidacji krzyżowej z partycjonowaniem zbiorku klasyfikowanych danych w podzbiory trenujące i walidujące (*linia 13* Alg. 4). Najlepszy klasyfikator wybierany jest na podstawie współczynnika dokładności (*ACC*, ang. *Accuracy*). Ocena klasyfikacji może zostać wykonana na podstawie miary dokładności z uwagi na równą dystrybucję zbioru trenującego.

W uproszczeniu Alg. 4 można zapisać w postaci 3 kroków wykonywanych dla kolejnych list cech F z listy list cech [F] wejściowych algorytmu:

1. Wyznacz transformatory w zależności od typu cechy
2. Wytrenuj klasyfikator dla każdego z algorytmów
3. Wybierz najlepszy klasyfikator porównując uzyskane wyniki dokładności (*ACC*)

Wyniki przeprowadzonych badań przedstawiłem w kolejnej sekcji, z uwzględnieniem miary średniej harmonicznej odzysku i precyzji (*F1*) wyznaczanej zgodnie z wzorami [102]:

$$Dokładność = \frac{TP + TN}{TP + TN + FP + FN} \quad (4.5)$$

$$Precyzja = \frac{TP}{TP + FP} \quad (4.6)$$

$$Odzysk = \frac{TP}{TP + FN} \quad (4.7)$$

$$F1 = \frac{2 * Odzysk * Precyzja}{Odzysk + Precyzja} \quad (4.8)$$

gdzie *TP*, *TN*, *FP*, *FN* odpowiadają wartościom wyniku klasyfikacji: prawdziwie pozytywna, prawdziwie negatywna, nieprawdziwie pozytywna, nieprawdziwie negatywna.

4.3.2. Wyniki eksperymentów

Opracowane podejście zilustrowane jest dla dwóch zadań klasyfikacji: typu zgłoszenia oraz kategorii komentarza. Przy ocenie modelu klasyfikacji wykorzystuję 6 implementacji algorytmów klasyfikacji dostępnych w bibliotece *sklearn* języka Python, takich jak: *Linear Support Vector (LSV), Naive Bayes for multinomial models (NB), Ridge regression (RR), Multi-layer Percepton* (percepton wielowarstwowy – sztuczna sieć neuronowa) (*MLP*), *Passive Aggressive (PA), k-nearest neighbors (KN)* dla obydwu zadań klasyfikacji.

W zadaniu klasyfikacji typu zgłoszenia wyróżniłem 4 etykiety: *User Story, Task, Sub-Task, Bug* wraz z 15 zestawami opracowanych cech. Zbiór trenujący składał się z 1200 zgłoszeń. Cechy, na których przeprowadzana była klasyfikacja, wybrałem na podstawie statystyk opracowanych zgodnie ze schematem przedstawionym w sekcji 3.1. Były to między innymi: treść tytułu zgłoszenia (T), Treść opisu zgłoszenia (D), sentyment tytułu zgłoszenia (TS), sentyment opisu zgłoszenia (DS), Liczba wystąpień słów specjalnych (|SW|), Liczba wystąpień adresów email (|Em|), Liczba odnośników zewnętrznych (|L|), Liczba załączników binarnych (|BA|), długość tytułu zgłoszenia (TL), długość opisu zgłoszenia (DL), procentowy udział słów technicznych (%TW). Pozwoliło mi to na oszacowanie wpływu wyznaczonych cech na wyniki klasyfikacji.

Wyniki klasyfikacji przedstawiłem w Tab. 13. Najwyższy współczynnik predykcji uzyskał zestaw cech: (T), (D), (|SW|), (|Em|) z dokładnością 0,936. Najniższy współczynnik dokładności uzyskany został natomiast dla algorytmu *PassiveAgressive* dla samej treści opisu zgłoszenia. Największy wpływ na dokładność klasyfikacji ma przeprocesowany tekst opisu zgłoszenia oraz treści tytułu zgłoszenia. Widoczna jest również potrzeba dobierania

dodatkowych cech klasyfikacji z dużą ostrożnością, ponieważ niektóre cechy powodują spadek *ACC* poniżej wartości otrzymanej wyłącznie dla treści tytułu i opisu zgłoszenia, np. zestaw cech T, D, TL, DL dla wszystkich rozpatrywanych algorytmów uzyskał współczynniki *ACC* i *F1* poniżej 89%.

Cechy	LSV	**MLP**	NM	RR	PA	KN
D	0.800;0.786	0.833;0.819	0.556;0.493	0.797;0.781	0.547;0.484	0.669;0.647
D, T, |SW|	0.919;0.919	0.936;0.934	0.881;0.874	0.917;0.915	0.906;0.901	0.825;0.809
D, T, SW|,|Em|	0.903;0.897	**0.936;0.935**	0.867;0.859	0.900;0.896	0.906;0.901	0.817;0.799
DL,|E|, |L|, |SW|, D, T	0.881;0.875	0.903;0.898	0.883;0.872	0.889;0.882	0.881;0.872	0.817;0.787
D, T, |L|, |BA|, |C|	0.917;0.914	0.919;0.917	0.872;0.866	0.906;0.900	0.886;0.880	0.797;0.776
D, T, %TW	0.914;0.913	0.922;0.921	0.886;0.884	0.911;0.908	0.908;0.907	0.789;0.773
D, T, DL	0.928;0.926	0.906;0.904	0.875;0.870	0.897;0.893	0.892;0.888	0.789;0.770

Tab. 13 Wyniki klasyfikacji typu zgłoszenia dla projektu P1 (wyniki podane w formacie: precyzja; średnia harmoniczna odzysku i precyzji)

Dla wszystkich algorytmów z zestawem cech %TW, |SW|, |TL|, |DL| współczynnik ACC jest zadowalający w zakresie 0,81-0,84. Jest to ciekawa obserwacja, biorąc pod uwagę fakt klasyfikacji zgłoszeń jedynie na podstawie wyznaczonych cech, bez potrzeby uwzględniania wyliczania metryki TF-IDF w procesie klasyfikacji, mogłoby to zmniejszyć czas potrzebny do trenowania klasyfikatora. Klasyfikacja taka mogłaby znaleźć swoje zastosowanie w miejscach, gdzie możemy pozwolić sobie na większy błąd, ale potrzebujemy uzyskać szybszy wynik predykcji. Wyznaczenie metryki TF-IDF dla dużych tekstów może być czasochłonne. Wyniki klasyfikacji dla projektów *open source* miały mniejszą dokładność (dla projektu MongoDB *ACC*=0,729), a wpływ dodatkowych cech na wynik klasyfikacji był widoczny.

Klasyfikacja zgłoszeń do 4 klas była stosunkowo prostym zadaniem, jednak może być również użyteczna w odniesieniu do innych celów, np. oceny jakości opisów, diagnostyki, gdzie wstępne przetwarzanie tekstu i używanie wyznaczonych cech zgłoszeń może mieć większe znaczenie. Warto zauważyć, że wysoka precyzja klasyfikacji może stanowić metrykę jakości opisów zgłoszeń. Klasyfikacja jakości opisów i diagnozowania wymaga włączenia aktorów projektu do stworzenia dużego zbioru trenującego.

W zadaniu klasyfikacji komentarzy wyróżniłem 4 kategorie: *Pozytywny, Odpowiedź, Pytanie, Poprawka*. To zadanie klasyfikacji jest bardziej wymagające niż klasyfikacja typu zgłoszenia. Jako zbiór trenujący klasyfikacji wykorzystałem 1200 komentarzy, które ręcznie sklasyfikowałem do założonych kategorii. Oceniłem wpływ dodatkowych cech na poziom

uzyskanej precyzji klasyfikacji takich jak: treść komentarza (przetworzona zgodnie z algorytmem x) (C), sentyment komentarza (CS), liczba odnośników do zmian w kodzie (|PR|), długość komentarza (|C|), liczba adresów email (|Em|), liczba odnośników zewnętrznych (|L|), liczba nazw klas (|CN|).

Cechy	LSV	MLP	NM	RR	PA	**KN**
C	0.825; 0.798	0.821; 0.793	0.806; 0.787	0.822; 0.794	0.802; 0.776	0.739; 0.707
C, CS, |PR|	0.881; 0.847	0.888; 0.873	0.881; 0.846	0.894; 0.864	0.874; 0.849	**0.904; 0.878**
C, |CODE|,|CN|	0.826; 0.798	0.829; 0.801	0.819; 0.802	0.829; 0.799	0.808; 0.783	0.673; 0.652
C, |CODE|,|CN|, CL, CS	0.868; 0.831	0.884; 0.858	0.828; 0.797	0.878; 0.848	0.865; 0.834	0.808; 0.759

Tab. 14 Klasyfikacja komentarzy projektu P1 (wyniki podane w formacie: precyzja; średnia harmoniczna odzysku i precyzji)

Dla wszystkich klasyfikatorów najlepszy wynik uzyskany został odpowiednio dla zestawu cech C, CS, |PR| dla projektu komercyjnego, z małą fluktuacją (0,05) pomiędzy uzyskanymi wynikami. Zadanie klasyfikacji komentarzy wykonałem również dla projektu *open source* Groovy, którego wyniki ilustruje Tab. 15. W tej klasyfikacji uwzględniłem dwie dodatkowe cechy: autor komentarza (CA) oraz liczbę znaków zapytania (|?|).

Cechy	**LSV**	MLP	NM	RR	PA	KN
C	0.643; 0.588	0.660; 0.614	0.653; 0.597	0.646; 0.593	0.621; 0.567	0.64; 0.560
C, CS,|PR|, |SW|	0.700; 0.651	0.703; 0.663	0.697; 0.650	0.696; 0.646	0.688; 0.638	0.644; 0.576
C, |PR|,CA	0.704; 0.660	0.697; 0.641	0.696; 0.645	0.689; 0.635	0.704; 0.652	0.670; 0.604
C, |PR|, |?|	**0.810; 0.785**	0.764; 0.721	0.805; 0.778	0.727; 0.688	0.740; 0.701	0.808; 0.776
C, CS, |?|	0.769; 0.739	0.754; 0.722	0.756; 0.730	0.686; 0.652	0.742; 0.710	0.714; 0.684

Tab. 15 Klasyfikacja komentarzy projektu Groovy (wyniki podane w formacie: precyzja; średnia harmoniczna odzysku i precyzji)

Najwyższy wynik dokładności oraz średniej harmonicznej klasyfikacji komentarzy projektu Groovy został osiągnięty dla zestawu cech C, |PR|, |?|. Interesujący jest także fakt, że połączenie tych cech poprawiło wynik precyzji dla wszystkich algorytmów. Cecha ta nie była widoczna w pierwszym zadaniu klasyfikacji. Predykcja bazująca jedynie na przetworzonej treści komentarza jest o gorsza niż wynik precyzji uzyskany z zastosowaniem dodatkowych cech (15% różnicy). Poprawa wyniku dokładności dla projektu P1 wynosiła 7,9%, widoczna jest zatem zasadność wykorzystywania dodatkowych cech w procesie klasyfikacji komentarzy zgłoszeń.

Wyższe wyniki dokładności (*ACC*) oraz średniej harmonicznej (*F1*) klasyfikacji komentarzy w projekcie komercyjnym mogą być uzasadnione lepszą znajomością projektu oraz ciągłym poprawianiem jakości dodawanych opisów przez aktorów projektowych. Zgłoszenia są regularnie weryfikowane oraz oceniane przez zewnętrzną organizację. Repozytorium podlega pełnemu przeglądowi przynajmniej raz do roku, w celu sprawdzenia możliwych usprawnień. Zespół projektu komercyjnego P1 cechuje się niskim poziomem fluktuacji (1-2 aktorów) na przestrzeni ostatnich lat, co przyczynia się do poprawy jakości raportowanych zgłoszeń. Wynik klasyfikacji (miara *ACC; F1*) może być traktowany jako miara jakości opisów zgłoszeń, co jest widoczne w przypadku projektu komercyjnego zarówno dla uzyskanych wyników klasyfikacji typów zgłoszeń oraz kategorii komentarzy.

4.4. Dyskusja

Zaprezentowane analizy zawartości tekstowej repozytoriów zgłoszeń potwierdziły, że składają się one w znacznym procencie ze słów języka naturalnego, które przemieszane są w różnym stopniu ze słowami spoza NL. Słowa te mogą dotyczyć słów specjalnych np. nazwy klas, modułów kodu, odsyłaczy, załączników itp. Istotną sprawą jest zidentyfikowanie takich tokenów oraz zastąpienie ich korespondującymi etykietami, aby ułatwić interpretację i zadania klasyfikacji. Proces ten wymaga pewnej wiedzy na temat przyjętych standardów w organizacji bądź projekcie. Przedstawione podejście wykorzystuje wyrażenia regularne, które mogą być zbudowane na podstawie obserwacji z manualnego przeglądu zgłoszeń. Aktywność ta może być powtarzana iteracyjnie w celu opracowania nowych wyrażeń. Proces ten jest stosunkowo prosty z uwagi na mały zbiór słów niesklasyfikowanych.

Repozytoria zgłoszeń zawierają także wiele opisów tekstowych, których analiza może znacząco wspomóc zespół projektowy. W pracy zaprezentowana została ocena implementacji 6 algorytmów klasyfikacji dla dwóch zadań klasyfikacji: typu zgłoszenia oraz klasyfikacja komentarzy. Automatyczna klasyfikacja typu zgłoszenia może znacząco zminimalizować niezbędny czas na wstępną analizę poprawności zgłoszenia. W sekcji 4.3.2. poświęconej dyskusji wyników klasyfikacji pokazuje zasadność użycia dodatkowych cech zgłoszenia generowanych przez dedykowany mechanizm ekstrakcji bazującej na treści tytułu oraz opisu zgłoszenia. Dodatkowe cech poprawiają dokładność klasyfikacji typu zgłoszenia. Precyzja klasyfikacji na podstawie dodatkowych klas może być pewnym wyznacznikiem jakości opisów zgłoszeń w repozytorium. W zadaniu klasyfikacji istotne jest także wybranie odpowiednich klas z zakresu badań. W projekcie P1 dla klas *User Story, New Feature, Bug, Task* miara ACC jest równia 0,93, jednak dla klas *User Story, New Feature, Bug, Sub-task* wynik był niższy,

ACC=0,846, z uwagi na wyraźne określenie typu Sub-task (mniej interesujące). W projekcie Groovy ACC wyniosło 0,729 (z uwagi na gorszą jakość opisów zgłoszeń).

Drugie zadanie klasyfikacji – klasyfikacja komentarzy – może stanowić rozwiązanie zauważalnego kłopotu w ustawianiu poprawnego statusu zgłoszeń w repozytoriach oraz dawać pogląd na możliwe kolejne kroki w obsłudze zgłoszenia. Wsparcie klasyfikacji zbiorem dodatkowych cech, znacząco poprawia dokładność klasyfikacji względem klasycznego podejścia bazującego jedynie na treści komentarza. Zbiór cech C, CS, |PR| dla projektu P1 poprawił wynik klasyfikacji o 7,9% z finalną dokładnością ACC=0,904. W projekcie Groovy zbiór cech C, |PR|, |?| uzyskał najwyższy wynik ACC=0,81 oraz ACC=0,664 dla niemodyfikowanej treści komentarza (poprawa o 15%).

Opracowany algorytm przetwarzania tekstu oraz detekcji słowników tekstów Alg. 1, Alg. 3 może zostać wykorzystany w innych tematach analiz jak np. wyszukiwanie podobieństw bądź klasteryzacji zgłoszeń, komentarzy. Może być połączony z innymi mechanizmami uczenia maszynowego (*machine learning*) lub eksploracji tekstów (*text mining*).

Klasyfikacja zgłoszeń z wykorzystaniem opracowanych algorytmów może pomóc także w rozwiązaniu problemu opisanego w sekcji 3.3.2. związanego z brakiem możliwości określenia ról poszczególnych reporterów zgłoszeń. W ramach prac badawczych wykonałem dodatkowy eksperyment klasyfikacji polegający na określeniu roli reportera bazując na opisach tworzonych przez niego zgłoszeń. Zbiór trenujący oraz testujący opracowałem ręcznie na podstawie projektu komercyjnego P1. W zadaniu klasyfikacji przypisywałem każdemu zgłoszeniu jedną z ról reportera: analityk, programista, właściciel produktu, tester. Najwyższy współczynnik osiągnął algorytm *MLP* z dokładnością równą 0,798 oraz średnią harmoniczną (F1) równą 0,799. Potwierdza to możliwość adaptacji opracowanego podejścia do innych zdań klasyfikacji. Istotnym problemem prac projektowych jest właściwe przypisanie zgłoszenia do odpowiedniej osoby (ang. *bug triaging*), w którym także możliwe było by użycie opracowanego mechanizmu klasyfikacji. Dodatkowej adaptacji wymagał by algorytm ekstrahowania cech zgłoszenia np. o komponenty określone w zgłoszeniu i połączenie ich z treścią zgłoszenia oraz dostępnymi komentarzami w celu automatycznego wyznaczenia najlepiej pasującej osoby.

Analiza słownikowa przedstawiona w sekcji 4.2. dostarcza również podstaw do wewnętrznych dyskusji zespołu w celu normalizacji stosowanego nazewnictwa, eliminując różnice wykazywane przez niektórych autorów tekstów. W zależności od przestrzeni analiz projektu profile słownikowe mogą być wyznaczone w różnych kontekstach np. analizy jakości

opisów, specjalnych funkcji lub możliwości diagnostycznych. Pewne cechy, wspomagające takie analizy, mogą być wyekstrahowane z opracowanych słowników. Perspektywy przedstawione w Rys. 2, Rys. 3 wskazują, że słownik w ujęciu miesięcznym opisów zgłoszeń jest nieduży, co ułatwia analizy. Profil agregacji słowników wskazuje jednak ich zmienność w perspektywie miesięcznej, dlatego w przypadku opracowywania klasyfikatorów z uczeniem nadzorowanym (ang. supervised machine learning), zasadne jest powtórzenie trenowania klasyfikatora w pewnych odstępach czasowych.

Przygotowany model klasyfikacji komentarzy wykorzystałem do badania dodatkowych ich cech w repozytorium, w tym ich sekwencji. Dla zobrazowania Tab. 16 uwzględnia sekwencję komentarzy (od 2 do 5 następujących po sobie komentarzy), dla których prezentuję czas od dodania zgłoszenia do pierwszego komentarza (T_1) oraz czas między pierwszym a ostatnim komentarzem sekwencji (T_L). Niektóre sekwencje wskazują na potrzebę dodatkowej interwencji managera bądź poprawę komunikacji w zespole, np. sekwencje *Poprawka, Pytanie* wskazują na brak odpowiedzi. Programista mógł pominąć dodanie odpowiedzi, ponieważ już wcześniej rozwiązał problem lub wynika to z jego zaniedbania. Kategoria *Odpowiedź* na początku sekwencji wskazuje na komentarz dotyczący zapytania w zgłoszeniu. Niektóre zgłoszenia tworzone są przez testerów w formie pytania czy odnaleziony przypadek jest błędem. Sytuacja ta może wskazywać także na braki w dokumentacji rozwiązania. Dla kilku zgłoszeń liczba komentarzy jest duża (nawet 10), co wpływa na różnorodność możliwych sekwencji komentarzy. W przypadku projektu *open soruce* (Groovy) średnia liczba komentarzy dla zgłoszenia nie przekracza 5.

Sekwencja komentarzy	T_1	T_L
Poprawka, Pytanie, Pozytywny	5d	5d
Poprawka, Poprawka, Pozytywny	5m	24d
Poprawka, Pytanie	7d	4,8d
Odpowiedź, Pytanie, Odpowiedź, Pytanie, Pozytywny	14m	62d

Tab. 16 Charakterystyka wybranych sekwencji komentarzy projektu P1

W projekcie komercyjnym kategorie dodawanych komentarzy są mocno skorelowane z typem autora komentarza. Komentarze typu *Poprawka*, dodawane są jedynie przez programistów, ponieważ to oni odpowiadają za zapewnienie połączenia między zgłoszeniem a zmianą w kodzie. Mniej oczywiste pozostają pozostałe kategorie które dodawane są przez wszystkie grupy aktorów. Pierwszy komentarz w zgłoszeniu jest zazwyczaj dodawany przez

programistę w szczególności w przypadku zgłoszeń błędów, gdzie informuje on o dostarczeniu poprawki lub zadaje pytanie dotyczące treści zgłoszenia. Zgłoszenia nowych funkcjonalności są komentowane w głównej mierze przez analityków i właściciela produktu oraz w małym stopniu przez testerów. Taka analiza komentarzy zgłoszeń w projektach *open source* jest niemożliwa z uwagi na brak możliwości ustalenia roli poszczególnych aktorów. Problem ten związany jest z pewnymi brakami w raportowaniu zgłoszeń (poprawna konfiguracja narzędzia) lub ograniczeniem dostępu dla osób nie związanych z projektem. Propozycję rozwiązania tego problemu omówiłem we wcześniejszym akapicie sekcji 4.4.

Dla różnych problemów klasyfikacji możemy połączyć eksplorację opisu zgłoszeń oraz powiązanych komentarzy z dodatkowymi cechami. Wydaje się mieć to istotny wpływ na ocenę jakości repozytoriów, bądź może być użyteczne w predykcji rozwiązania zgłoszenia. Analiza może dotyczyć różnych zgłoszeń, komentarzy specyficznych do typów zgłoszeń, okresów analizy lub zaangażowanych reporterów.

Szczegółowa analiza pól opisowych zgłoszeń wykazała ich złożoność oraz potrzebę pogłębienia eksploracji występujących słowników. Opisy i komentarze zgłoszeń składają się z szerokiego zakresu słów NLW, żargonu technicznego, skrótów, mieszanki słów NLW reprezentujących nazwy obiektów programowych oraz wielu innych fraz (odnośniki, nazwy załączników). Przedstawione schematy analiz rozpoczynają się od opracowania słowników oraz zastąpienia fraz odpowiednimi etykietami. Zadania klasyfikacji mogą zostać wsparte cechami semantycznymi weryfikowanych tekstów, co potwierdziłem we wcześniejszej dyskusji. Konkretne metody klasyfikacji mogą zostać poprawione o dodatkowe cechy wyznaczane na podstawie specyfiki projektów oraz specjalnych słów występujących w opisach. Metody te mogą zostać dodatkowo poprawione dzięki włączeniu analiz kontekstowych i korelacji.

Wartościowa byłaby klasyfikacja jakości opisów zgłoszeń powiązana z ich oceną przez aktorów projektu. W szczególności w zakresie zrozumiałości opisu, łatwości lokalizacji błędu. Do tego celu można wykorzystać przedstawioną metodykę, przy czym przygotowanie zbioru testującego wymaga zaangażowania odpowiednich aktorów np. analityków, programistów do etykietowania zbioru zgłoszeń. Tu też ten problem można rozpatrywać dla różnych typów zgłoszeń oddzielnie np. błędów, nowych funkcjonalności. Podobnie można je odnosić do różnych grup aktorów w celu wyodrębnienia osób generujących „gorsze” opisy. Wyniki analiz klasyfikacji mogą również być przydatne w analizie obsługi zgłoszeń rozpatrywanej w rozdziale 5.

5. Analiza obsługi zgłoszeń

Ewaluacja procesu obsługi zgłoszeń może być przeprowadzona na dwóch poziomach: 1) ogólnym, w którym istotne są 3 metryki: czas obsługi zgłoszenia, zakres nierozwiązanych zgłoszeń i rozkład zmian kodu względem zgłoszeń (sekcja 5.1.), lub 2) szczegółowe (drobnoziarniste) ukierunkowane na śledzenie następujących po sobie stanów obsługi zgłoszenia. Dla przeprowadzenia takich analiz opracowałem model graficzny IHG wraz z szeregiem profili i algorytmów, których szczegółowy opis przedstawiony jest w sekcji 5.2. Model ten wspiera śledzenie ścieżek obsługi zgłoszeń opisanych w sekcji 5.2.4. i 5.2.6.

5.1. Ewaluacja obsługi zgłoszeń – poziom ogólny

Analiza procesu obsługi zgłoszeń w perspektywie ogólnej (gruboziarnistej) może uwzględniać następujące aspekty:

- Przepływ rejestrowanych zgłoszeń w czasie (wykresy okresowe i skumulowane)
- Charakterystyki czasowe obsługi zgłoszeń
- Efektywność rozwiązywania napływających zgłoszeń
    - Rozkład w czasie (profil bezwzględny) zgłoszeń oczekujących na rozwiązanie
    - Identyfikacja odłożonych zgłoszeń krytycznych (tzw. dług błędów)
- Charakterystyka implementowanych zmian kodu

Wyniki takich analiz można przedstawić w formie odpowiednich wykresów lub w formie tabelarycznej. Analizy te można również przedstawić w odniesieniu do poszczególnych typów zgłoszeń, ich priorytetów, reporterów itp. Poniżej ograniczyłem się do wybranych przykładów ilustrujących obserwacje w oparciu o zgłoszenia JIRA przetworzonych przez opracowaną aplikację *IssueAnalyzerTool* i dedykowane skrypty eksploracji danych (*data mining*).

Dla każdego zgłoszenia możemy zmierzyć czas jego obsługi uwzględniając czas od dodania zgłoszenia w repozytorium do czasu jego zamknięcia (ustawienia stanu terminalnego). Stan terminalny zgłoszenia wymaga komentarza, teoretycznie stanem terminalnym zgłoszenia powinien być jeden stan np. *Closed* (zamknięty) bądź jego odpowiedniki. W wielu projektach występuje jednak problem z procesowaniem zgłoszeń, najczęściej w projektach *open soruce*, gdzie zgłoszenia nie są procesowane do stanu *Closed,* a kończą obsługę najczęściej w stanie *Resolved_X* (rozwiązany, gdzie X specyfikuje sposób rozwiązania zgłoszenia np.: *Fixed* – naprawienie kodu, *Rejected* – odrzucenie zgłoszenia).

W przypadku, kiedy zgłoszenie nie trafia do stanu *Closed* musimy uznać stan *Resolved_X* za terminalny. Innym napotkanym problemem w weryfikowanych repozytoriach był brak osiągnięcia przez zgłoszenie stanu terminalnego (uwzględniając status *Resolved_X*) w analizowanym zakresie czasu. Dlatego statystyki (Tab. 17) ujęte są w dwóch wariantach: I_A - uwzględniający jedynie zgłoszenia zamknięte, I_B uwzględniający zgłoszenia nie będące w stanie terminalnym. Zgłoszenia zaprezentowane w tabeli Tab. 17, Tab. 18, Tab. 19 uwzględniają typy reprezentujące zgłoszenia błędów (*Bug*) oraz nowych funkcjonalności (dla open source - *New Feature, Improvement*, dla P1 - *New Feature, User Story, Task*).

Projekt	N	AVG	MAX	Q2	Q3
Mdb I_A	11675	45,2	358,2	28,4	52,3
Mdb I_B	1760	242,5	499,6	232,2	326,1
P1 I_A	2059	45,4	226,2	30,0	48,9
P1 I_B	319	52,0	80,2	48,8	59,9
Lo I_A	2343	153,5	1721,3	40,7	125,7
Lo I_B	764	7,5	26,2	2,8	7,0

Tab. 17 Charakterystyka ogólna obsługi zgłoszeń w projektach MongoDB (Mdb), P1, Log4J2 (Lo) dla dwóch wariantów zgłoszeń zamkniętych (I_A) i zgłoszeń nie rozwiązanych (I_B - bez ustawienia stanu terminalnego)

Opisaną charakterystykę obserwacji przedstawiłem w Tab. 17 dla projektów: Mdb – MongoDB, P1 – projekt komercyjny, Lo – log4J2. Wartości w kolumnie N reprezentują liczbę zgłoszeń. Kolumny AVG, MAX, Q2, Q3 odnoszą się do czasu w dniach, określając odpowiednio wartości średnią, maksymalną, kwartyl Q2 i kwartyl Q3. W projektach Mdb i P1 zgłoszenia, których procesowanie nie zostało zakończone (I_B) charakteryzują się wysokim czasem obsługi. Pozostałe raporty, które zostały przeprocesowane do stanu terminalnego I_A, przetwarzane były z medianą odpowiednio 28,4 dni (Mdb) i 30 dni (P1).

Inną perspektywą oceny procesu jest liczba zarejestrowanych zgłoszeń na konkretny moment w czasie oraz liczba zamkniętych zgłoszeń. Tab. 18 i Tab. 19 przedstawiają te statystyki w perspektywie 10 lat dla projektów: Mad – MariaDB, Sak – Sakai, Gr – Groovy, Lo – Log4J2, Lu – Lucene, Ar – Arrow, komercyjny P1.

Projekt	1	2	3	4	5	6	7	8	9	10
Mdb	3,8	6,0	5,4	6,8	7,1	6,4	7,2	10,3	10,0	17,6
Mad	7,6	10,6	12,4	15,6	18,1	23,0	27,3	32,1	37,7	48,5
Sak	2,3	5,2	10,2	19,3	23,8	23,2	20,2	24,4	32,5	40,3
Gr	9,3	12,9	18,6	18,2	29,8	21,0	20,4	20,5	19,9	22,3
Lo	4,5	16,7	22,4	21,4	24,9	30,6	36,7	32,7	34,9	40,6
Lu	17,8	21,7	23,5	21,7	24,1	20,7	32,0	25,6	30,8	32,7
Ar	0	0	0	2,5	2,0	5,5	8,0	12,7	20,3	35,2
P1	0	0	0	0	16,6	5,9	6,7	7,1	6,2	12,5

Tab. 18 Profil procentowy nierozwiązanych zgłoszeń w kolejnych 10 latach trwania projektu

Projekt	1	2	3	4	5	6	7	8	9	10
Mdb	123	330	543	795	1021	1271	1557	1992	2458	3263
Mad	85	242	429	704	1146	1740	2424	3242	4160	5309
Sak	32	104	223	600	891	1201	1461	1829	2340	2870
Gr	55	145	228	280	388	476	543	625	712	786
Lo	14	75	137	234	317	410	482	547	638	757
Lu	122	270	430	556	660	749	882	976	1103	1190
Ar	0	0	0	9	38	145	378	780	1469	2366
P1	0	0	0	0	84	250	374	534	671	968

Tab. 19 Liczba zgłoszeń oczekujących na rozwiązanie w kolejnych 10 latach trwania projektu

Hipotetycznie trend liczby nierozwiązanych zgłoszeń powinien być malejący, z uwagi na coraz lepsze zrozumienie specyfiki projektu przez zespół, jednak hipoteza ta nie została potwierdzona przez dane. Teoretycznie założenie to mogłoby być prawdziwe w przypadku projektów o stałej liczbie użytkowników i ustalonej funkcjonalności, jednak weryfikowane projekty *open source* zyskiwały coraz większą popularność i nowe funkcjonalności. Trend jest wzrostowy i świadczy o położeniu projektu w fazie rozwoju. Tendencja malejąca widoczna była w projekcie komercyjnym P1, jednak w ostatnich latach nastąpił przyrost nierozwiązanych zgłoszeń, związane jest to bezpośrednio z nowymi funkcjonalnościami i dynamicznym rozwojem projektu. Normalną sytuacją mogłyby być chwilowe anomalie wzrostu

nierozwiązanych zgłoszeń w projektach np. w ujęciu miesięcznym. Jednak sukcesywny przyrost nierozwiązanych zgłoszeń jest wysoce niepożądaną sytuacją.

W ocenie projektów wartościowy jest profil rozkładu priorytetów otwartych zgłoszeń. Najwięcej krytycznych (Priorytet *Blocker*) nierozwiązanych zgłoszeń pojawia się w projekcie komercyjnym jest to średnio 5 zgłoszeń na rok w ostatnich 5 latach z fluktuacją między 2 a 10. Spośród projektów OS alarmujące wydają się zgłoszenia projektu Lo (Log4J2), w którym nie zamykano średnio 2 zgłoszeń rok do roku w ostatnich 6 latach z fluktuacją 1-5. W ostatniej części badanego okresu widoczny jest wzrost liczby nierozwiązanych zgłoszeń o priorytecie *Blocker* w projektach Mdb - 10, Sak - 2, Lo - 10, Ar – 5. Niezależnie od projektu liczba nierozwiązanych krytycznych zgłoszeń jest jednak znikoma (średnio poniżej 1%), co można ocenić pozytywnie. Całkowita liczba nierozwiązanych zgłoszeń powinna zostać zweryfikowana przez zespół i świadczy o problemach w rozwiązywaniu zgłoszeń w projektach *open source*, gdzie średnia liczba nierozwiązanych zgłoszeń dla badanych lat wynosiła odpowiednio 8,1% (Mdb), 23,3% (Mad), 20,1% (Sak), 19,3% (Gr), 26,5% (Lo), 25,1% (Lu), 8,6% (Ar). Dla porównania, średnia wartość nierozwiązanych zgłoszeń rok do roku w projekcie komercyjnym P1 wynosiła jedynie 5,5%.

Proces rozwiązywania błędów jest istotny w czasie rozwoju i utrzymywania oprogramowania. W praktyce jednak potrzeby biznesowe (nowe funkcjonalności), w szczególności terminarz udostępnienia nowych funkcjonalności może stać w konflikcie z rozwiązaniem wszystkich błędów. W takich przypadkach zespół stawiany jest przed podjęciem decyzji – wstrzymać pracę i poświęcić czas na rozwiązanie wszystkich błędów czy skupić się jedynie na najbardziej istotnych i dostarczyć rozwiązanie w terminie. Pytanie to można przenieść na wyższy poziom: jakimi priorytetami powinien kierować się zespół. Czy ważniejsza w dostarczanym oprogramowaniu jest jakość czy liczba nowych funkcjonalności możliwych do dostarczenia w tym samym czasie? Najczęściej w przypadku projektów komercyjnych oraz zmiennych priorytetów biznesowych dochodzi do kompromisu, tzn. zespół stara się poprawić błędy w jak najkrótszym czasie, niejednokrotnie stosując nieoptymalny kod (np. dodawanie nowych warunków obsługujących specyficzny przypadek opisany w błędzie zamiast generycznego rozwiązania) oraz pomija zgłoszenia o nikłej istotności. Postępowanie takie prowadzi jednak do budowania skomplikowanego, nieoptymalnego kodu (ang. spaghetti model) [103], którego rozwój i utrzymanie obarczone jest dużym kosztem (koszt rozumiany jako czas poświęcany przez zespół do implementacji nowej funkcjonalności bądź poprawki błędu). Literatura definiuje ten problem poprzez pojęcie długu technicznego (ang. *technical*

debt), które zostało opisane w literaturze przez Ward-a Cunningham-a (1992) jako: „*Shipping first time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.*" [104]. Dług techniczny odnosi się jednak do kodu, który nie jest napisany zgodnie z najlepszymi praktykami, interesujące jest także połączenie pojęcia długu technicznego ze zgłoszeniami błędów szczególnie tymi otwartymi.

W literaturze autorzy podejmują próby zdefiniowania pojęcia długu defektów (ang. *defect debt*) [105], jako jeden z czynników długu technicznego. Jednak analizy skupiają się na określeniu pochodzenia takiego długu oraz jego klasyfikacji, np. jako zgłoszenia duplikatów. Źródło pojawienia się długu defektów jest ciekawym aspektem analiz. W pracy definiuję ten obszar jak dług błędów, ponieważ wartościowsze dla projektów jest odkrycie źródła problemu w procesowaniu zgłoszeń błędów. Pojęcie długu błędów określa aktualną liczbę zgłoszeń błędów, które nie znajdują się w stanie terminalnym czyli nadal są w trakcie procesowania. Dług błędów nie może być jednak określony jako podtyp długu technicznego, ponieważ zgłoszenia te nie zostały jeszcze w pełni przeprocesowane, a tym samym projekt nie został jeszcze obarczony możliwymi uproszczeniami technicznymi na potrzeby szybkiego rozwiązania błędu. Pojęcie to może jednak stanowić źródło długu technicznego. Dług błędów może prowadzić do długu technicznego oraz wzrostu ryzyka wynikającego z niepoprawienia krytycznego błędu, który może rzutować na stabilność całego oprogramowania.

W analizowanym projekcie komercyjnym sukcesywnie dodawane były nowe błędy o nikłej istotności, od początku weryfikowanego okresu. Każdego miesiąca wzrastała ich liczba do kulminacyjnego momentu, w którym liczba nierozwiązanych zgłoszeń wynosiła 366. Otwarte błędy pojawiały się jednak z opóźnieniem względem daty rozpoczęcia projektu, tzn. "Dług błędów" występuje od 5 miesiąca weryfikowanego okresu. Przez długi okres czasu mediana pomijanych otwartych raportów z każdego miesiąca utrzymywała się na poziomie 1 zgłoszenia błędu (które nie było zamykane w każdym miesiącu), jednak po fazie stabilizacji średnio 4 błędy/miesiąc przez 8 miesięcy, następnie nastąpił nagły wzrost długu otwartych błędów. Odpowiednio 13(110), 16 (146), 15 (99), 21 (79), 32 (147), 45 (143), 47 (146), 63 (141), 55 (88), dla każdego z kolejnych miesięcy, w nawiasie podaję liczbę wszystkich błędów zgłoszonych w danym miesiącu. Niestety dane te nie wykazują żadnej korelacji, ponieważ we wcześniejszym okresie dług błędów nie rósł tak szybko, pomimo podobnej łącznej liczbie tworzonych błędów w każdym miesiącu. Dzięki znajomości projektu jestem w stanie uzasadnić

to możliwym wypaleniem programistów, którzy po pewnym czasie tracą zapał do rozwiązywania podobnych błędów – które wielokrotnie są żmudne w identyfikacji wadliwej części kodu, a ich rozwiązanie jest mało widoczne w perspektywie wysokopoziomowej projektu. Potwierdza to potrzebę wykonywania rotacji ról w zespołach deweloperskich tak, aby każdy z programistów miał możliwość zdobywania nowych umiejętności w różnych obszarach tworzonej aplikacji, ale również mógł zmieniać zakres swoich prac nie tylko poprawiając przez cały czas błędy z jednego obszaru aplikacji.

Rosnący dług błędów niesie dodatkowe ryzyko dla projektu wynikające z błędnej ocena ważności błędu, który nie będzie rozwiązany w stosownym czasie. Jeżeli organizacja/grupa projektowa akceptuje tworzenie długu błędów ze zgłoszeń o małej istotności, trafiają one na koniec listy zgłoszeń do weryfikacji. Może to spowodować propagacją błędu o źle określonym priorytecie do środowiska produkcyjnego. Pseudokod algorytmu detekcji długu błędów wraz ze wzrostami w różnych okresach czasu przedstawiłem w Alg. 5.

Alg. 5 operuje na 4 danych wejściowych: data początku weryfikowanego okresu (*startDate*), data końca weryfikowanego okresu (*endDate*), interwał czasowy (*I* - np. miesiąc bądź sprint) oraz próg powyżej którego liczba nierozwiązanych błędów będzie zliczana (*T*). Algorytm wykorzystuje funkcję pomocniczą *get_unresolved_issues(startDate,endDate)*, która generuje zapytanie do JIRA o nierozwiązane zgłoszenia błędów w określonym oknie czasowym (*linia 6*). W opracowanym podejściu pomijam jednorazowe wysokie wartości długu błędów (*linie 10-16*), ponieważ z perspektywy projektowej anomalia ta nie stanowi problemu jeżeli jest ona jednorazowa. Problemem jest jednak stały wzrost liczby nie rozwiązanych zgłoszeń w następujących po sobie miesiącach. W uproszczeniu algorytm zapisany w pseudokodzie Alg. 5 dla każdego okna czasowego (*I*) w zakresie dat *startDate* – *endDate* można opisać krokami:

1. Sprawdź czy liczba błędów przekracza próg *T* (*linie 7-8*)
2. Zbuduj listę wzrostów długu błędów z uwzględnieniem progu istotności (*T*) oraz pominięciem jednorazowych miesięcy z długiem błędów (*linie 7-17*)

Zjawisko "Długu błędów" występuje w weryfikowanych projektach typu *open source,* podobnie jak w projekcie komercyjnym.

```
input: start date (startDate) and end date (endDate) of verification time
period, interval (I) in which unresolved bugs will be checked, threshold (T)
of bug which can be skipped
output: list of time intervals (GR) with raising bug debt identified as tuple
with month and list of unresolved bugs created in month
```

```
1 function bug_debt_growth(startDate, endDate, I, T):
2    GL = new empty list
3    beginDate = startDate
4    TMP = new empty list
5    WHILE(beginDate + I < endDate) DO:
6        UI = get_unresolved_issues(begindate, beginDate + I):
7        IF TMP == empty OR (UI > T AND len(TMP.last[1]) < UI) THEN
8            TMP.add(Tuple(beginDate.getMonth(), UI)
9        ELSE:
10           IF tmpList.size() >= 2 THEN
11               GL.add(tmp)
12               TMP = new empty List
13           ELSE:
14               TMP = new empty List
15           ENDELSEIF
16       ENDELSEIF
17       beginDate = beginDate + interval
18    ENDWHILE
19    return GL
20 endfunction
```

Alg. 5 Pseudokod detekcji długu błędów wraz ze wzrostami

Poniżej prezentuję rozważania bazujące na wynikach otrzymanych z Alg. 5. Wzrost długu pojawia się z opóźnieniem względem rozpoczęcia każdego z projektów oraz daty zgłoszenia pierwszego błędu. Problemy z rozwiązywaniem zgłoszeń pojawiają się odpowiednio po 1 miesiącu dla AirFlow, 2 miesiącach dla Arrow, 36 miesiącach dla Groovy, 18 miesiącach dla Log4J2, 15 miesiącach dla Lucene, 9 miesiącach dla MariaDB, 29 miesiącach dla MongoDB, 5 miesiącach dla Sakai. Dodatkowo, zauważalna jest tendencja wzrostowa liczby nowo tworzonych i nierozwiązanych zgłoszeń błędów wraz z upływem czasu od początku każdego z projektów. Dług błędu koreluje z liczbą wszystkich zgłoszeń oraz przyrostem długu poprzez wektor <***całkowita wielkość dług błędów**, łączna liczba zgłoszonych błędów w projekcie, średnia wartość przyrostu długu błędów w miesiącu*>. Dla badanych projektów wartość tego wektora wynosiła: AirFlow <***796**;3042;12*>, Arrow <***476**;4692;7*>, Groovy <***539**;5704;3*>, Log4J2 <***409**;1595;3*>, Lucene <***638**;3853;3*>, MariaDB <***4828**;19259;31*>, MongoDB <***907**;24619;6*>, Sakai <***5238**;28198;26*>. Wektor korelacji pozwala ocenić procentowy udział długu błędów (opóźnionych zgłoszeń) względem wszystkich zgłoszeń oraz jego średni przyrost w miesiącu. W każdym projekcie wartość alarmująca będzie różna, zależnie od ustaleń organizacji oraz poziomu akceptowalnego ryzyka wynikającego z nierozwiązanych zgłoszeń.

Sytuacja narastającego długu błędów może być wywołana wieloma czynnikami takimi jak: spadek efektywności zespołu, wzrost liczby zgłoszeń nowych funkcjonalności które zostały dostarczone w krótkim czasie poprzedzającym wzrost długu błędów, problemy z alokacją zadań w zespole lub duża fluktuacja zespołu skutkująca obniżeniem wydajności prac.

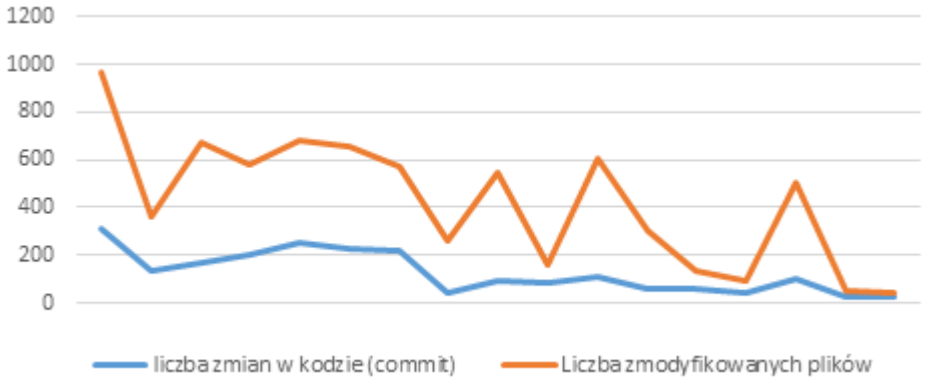
Dane wyjściowe generowane przez Alg. 5 mogą być przetwarzane ręcznie przez eksperta projektu w celu oceny ich istotności lub mogą zostać zweryfikowane przez algorytmy klasyfikacji (*machine learning*). Kroki detekcji podejrzanych zgłoszeń błędów których rozwiązanie zostało odłożone na późniejszy (nieokreślony termin) można uprościć do postaci:

1. Identyfikacja listy znaczącego nieliniowego wzrostu opóźnionych błędów (Alg. 5 zbudowanie listy D_P, tych błędów)
2. Selekcja podejrzanych zgłoszeń z listy D_P z wykorzystaniem metod analiz tekstów (*text mining*)

W kroku 2. wartościowe jest zastosowanie klasyfikacji na podstawie słowników przedstawionych w sekcji 4.2. Możliwe do wyznaczenia są słowa kluczowe identyfikujące zgłoszenia krytyczne z wysoką dokładnością z długu błędów. Przykładowo dla projektu Arrow, w którym pełen słownik NLW, FW, NCW, CCW składał się łącznie z 1548 słów wyznaczyłem ręcznie słowa kluczowe (na podstawie manualnych obserwacji): *incomplite, ignoring, loss, errored, error, timeout, frozen, elapsed, vewlogs, overloaded, adapter, halted, illegal, hangs, killed, unspecified.* Bazując na ich wystąpieniach w opisach zgłoszeń generowanych z algorytmu Alg. 5, uzyskałem 79% poprawności predykcji zgłoszeń krytycznych. Dla Projektu Log4J2 z 418 słów pełnego słownika wyznaczyłem 28 słów kluczowych. Predykcja na ich podstawie dała poprawność na poziomie 72%. Innym podejściem może być zastosowanie klasycznych metod uczenia maszynowego (*machine learning*). Przeprowadziłem taki eksperyment, z wykorzystaniem zbioru trenującego składającego się z opisów zgłoszeń o najwyższym priorytecie (*Blocker*). Dokładność predykcji krytycznych zgłoszeń z długu błędów z wykorzystaniem algorytmu klasyfikacji *k-nearest neighbors* znajdowała się w zakresie 72% - 90% dla 8 testowanych projektów - 7 *open source* (Airflow, Groovy, Log4J2, Lucene, MariaDB, MongoDB, Sakai) i projektu komercyjnego (P1). Najwyższa wartość dokładności osiągnięta została dla projektu P1, co potwierdza wysoką jakość opisów zgłoszeń.

Problemy korelacji zgłoszeń ze zmianami kodu były badane we wcześniejszych pracach prowadzonych w zakładzie ZOiAK Instytutu Informatyki, np. w pracy [82]. Dla porównania

podałem wyniki dla projektu P1 (Rys. 4). Temat ten nie był przedmiotem szczegółowych badań w ramach rozprawy.



Rys. 4 Korelacja liczby zmian w kodzie (*commit*) ze zmodyfikowanymi plikami w Projekcie P1 (linia pozioma określa kolejne miesiące dla okresu 17 miesięcy)

Hipotetycznie wysoka liczba zmienionych plików w jednym *commicie* może sugerować potrzebę podzielenia zadań na mniejsze części. W projekcie komercyjnym średnio na 1 *commit* modyfikowane były 3 pliki. Średnia liczba zmienionych plików jest niska z uwagi na wysoki poziom granulacji prac projektowych, który wykorzystywany jest w metodologii *SCRUM*. Każe z zadań powinno być podzielone do jak najmniejszych części co gwarantuje jego dobrą estymację i szybkie dostarczenie. Wykres Rys. 4 różni się od obserwacji przedstawionych w [82], w których z upływem czasu histogram był malejący. Może to być związane z finalizacją analizowanych przez autora projektów, które przechodzą w fazę utrzymania.

Istotne jest, aby zaznaczać, że nie wszystkie zgłoszenia wymagają zmian w kodzie. W Projekcie P1 tylko 49,8% zgłoszeń wymagało modyfikacji kodu dla zgłoszeń typu (*Bug, User Story, Task, New Feature*). Biorąc pod uwagę same zgłoszenia błędów, 51,7% z nich wymagało poprawek kodu aplikacji, w projekcie MongoDB procent ten był równy65,7%. Pozostałe zgłoszenia rozwiązane zostały przez zmiany konfiguracji, bądź nie odnoszą się bezpośrednio
do zmian w kodzie. Przykładem takiego typu zgłoszenia jest *New Feature*, który agreguje zgłoszenia typu *User Story*, przez co nie ma bezpośredniego odniesienia do kodu aplikacji.

5.2. Grafowy model obsługi zgłoszeń (IHG)

Proces obsługi zgłoszeń może składać się z różnych statusów, w zależności od określonego przepływu w projekcie bądź organizacji. Niezależnie jednak od nazewnictwa stanów oraz finalnego sposobu rozwiązania, proces ten wymaga stałego monitorowania efektywności.

W tym celu opracowałem koncepcje grafów obsługi IHG. Model IHG jest istotnym rozszerzeniem grafów PHG z mojej pracy magisterskiej [12], które były wykorzystywane w późniejszych pracach Instytutu np. [10, 11, 13, 14] w tym z moim udziałem. Rozpatrywano tam raporty błędów na poziomie ogólnym. Model IHG wspierany jest przez opracowane oryginalne algorytmy analizy powiązane z wprowadzonymi profilami strukturalnymi i statystycznymi miarami obejmującymi różne aspekty procesów obsługi zgłoszeń w kontekście stanów i ścieżek grafu. Uwzględniono tu podstawowe typy zgłoszeń oraz ich atrybuty opisane w rozdziale 3.

Graf obsługi zgłoszeń (IHG) dla projektu P definiuję jako:

$$IHP(P) = \{V(P), E(P), \delta, \gamma\} \,||R \qquad (5.1)$$

gdzie V(P) – zbiór wierzchołków grafu opisującego stany (fazy) obsługi zgłoszenia, E(P) – zbiór ukierunkowanych krawędzi grafu opisujących zmiany stanu, δ oraz γ – funkcje określające odpowiednio cechy wierzchołków (stanów) i krawędzi (zmiany stanów) grafu. Możemy użyć zbioru tych funkcji do określenia nazwy stanu, liczby zgłoszeń skorelowanych z tym stanem lub krawędzią, parametr czasowy itp. Ograniczenie R określa zakres modelu grafu ograniczającego zbiór uwzględnionych typów zgłoszeń (np. błędów, nowych funkcjonalności). Funkcja γ nawiązuje do przepływu zgłoszenia, które można opisać przykładowymi wzorami (analogicznymi do podanych w [14]):

a) $$\bigvee_{e_{ij} \epsilon E} \quad \gamma_1(e_{ij}) = n_{ij} \qquad (5.2)$$

b) $$\bigvee_{e_{ij} \epsilon E} \quad \gamma_2(e_{ij}) = \frac{n_{ij}}{N} \times 100\% \qquad (5.3)$$

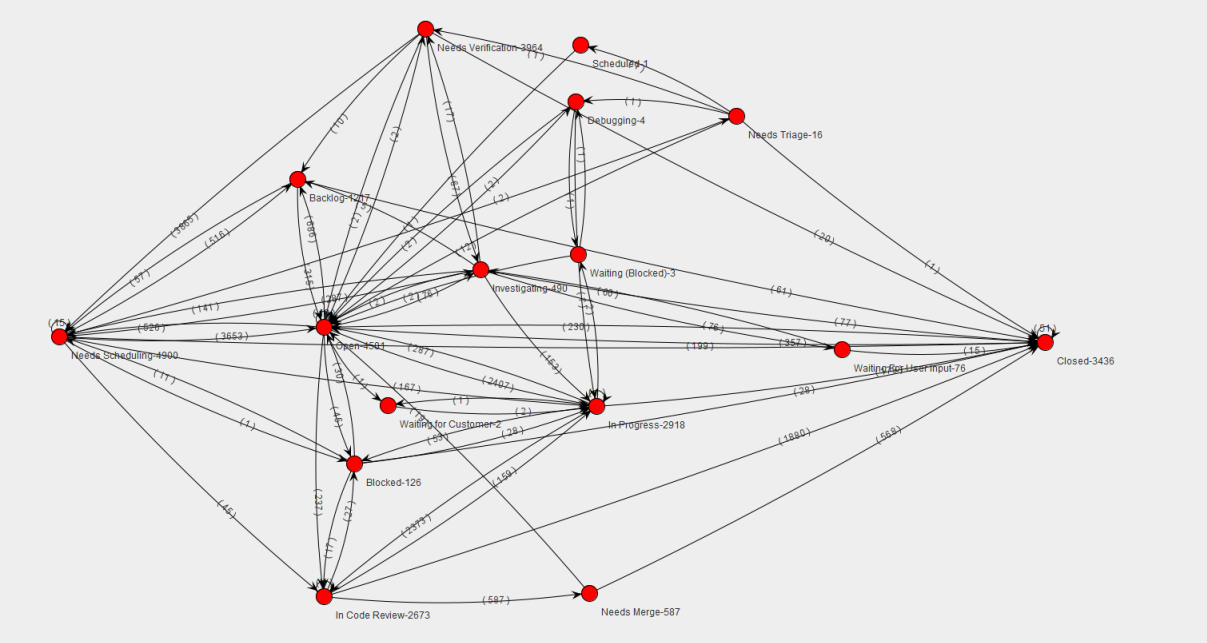
c) $$\bigvee_{e_{ij} \epsilon E} \quad \gamma_3(e_{ij}) = \frac{n_{ij}}{\sum_{e_{ij} \epsilon\, out(V_i)} n_{ij}} \times 100\% \qquad (5.4)$$

gdzie: n_{ij} – liczba zgłoszeń skorelowanych z krawędzią e_{ij}, wychodzących z wierzchołka v_i i wchodzących do następującego wierzchołka v_j; $v_i, v_j \in V(P)$; $e_{ij} \in E(P)$; N – liczba wszystkich weryfikowanych zgłoszeń; out(v_i) – zbiór wszystkich krawędzi wychodzących z wierzchołka i. Funkcja γ_1 określa absolutną liczbę zgłoszeń. Funkcja γ_2 określa względną liczbę zgłoszeń w odniesieniu do całkowitej liczby zgłoszeń ujętych w grafie (względna przepustowość). Funkcja γ_3 określa względną liczbę zgłoszeń obsłużonych względem liczby raportów obsłużonych przez źródłowy stan rozpatrywanej krawędzi. Wykorzystując różne formy tych funkcji, możemy skoncentrować się na wybranych aspektach podczas wizualizacji grafu (perspektywa globalna lub lokalna).

Zależnie od celu analizy możemy nałożyć określone ograniczenia (R) na rozpatrywane zgłoszenia, w szczególności:

- Klasy zgłoszeń - zgłoszenia określonego typu, priorytetu, istotności, stworzonych przez wybranego użytkownika bądź grupę użytkowników
- Czas obsługi - zgłoszenia stworzone do lub od znacznika czasu, stworzone w zdefiniowanym przedziale czasu, rozwiązane do znacznika czasu lub w zakresie czasu, zgłoszenia powiązane z czasem trwania sprintu, wersją programu

Ponadto możemy generować zredukowane grafy poprzez filtrację wierzchołów (stanów) oraz krawędzi (przejść zgłoszeń pomiędzy stanami) o pomijalnej istotności określającej odpowiednio liczbę obsłużonych zgłoszeń przez stan, liczbę zgłoszeń, które przeszły przez krawędzie np. poniżej 1%. Używając różnych funkcji, możemy skoncentrować się na różnych aspektach wizualizacji grafów (globalnej bądź lokalnej).



Rys. 5 Graf obsługi zgłoszeń projektu MongoDB z ograniczeniem R – typ zgłoszenia Improvement

Dla ilustracji Rys. 5 przedstawia pełen graf IHG dla zgłoszeń 4257 zgłoszeń (3155 zgłoszeń zamkniętych i 1102 zgłoszeń otwartych) typu *Improvement* projektu MongoDB, który składa się z 16 stanów. Redukcja grafu do poziomu istotności powyżej 10% (stanów oraz wierzchołków) ogranicza liczbę stanów do 8: *Needs Verification, Needs Scheduling, Backlog, Open, In progress, In Code Review, Needs Merge, Closed.* Dla porównania, pełen graf obejmujący zgłoszenia typu *Bug, Improvement, New Feature* zawiera 21 stanów. Opracowane narzędzie *IssueAnalyzerTool* generuje graf IHG w przyjaznej dla użytkownika formie

graficznej (Rys. 5) z wykorzystaniem biblioteki Jung [106]. Więcej przykładów analiz grafów podano w sekcji 5.3.2. i załączniku 8.2.

Algorytmy analizy grafów IHG są zaadoptowane do ich złożoności określonych parametrami: |V| – liczba wierzchołków (używane zamiennie z określeniem stan), |E| – liczba krawędzi (rozmiar grafu), d(G) = |E|/(|V|*(|V|-1)) – gęstość grafu, indeg(V) i outdeg(V) – liczba wejściowych i wyjściowych krawędzi wierzchołka. Generując grafy dla wielu projektów *open soruce* i komercyjnych, stwierdziłem, że ich złożoność jest relatywnie niska w porównaniu z grafami klasycznymi [26]. W większości przypadków wielkości rozmiaru grafu |V| jest w zakresie 25-48, indeg(v) 1-6; outdeg(V) 1-7, gęstość grafu 0,15-0,43. Wartości te znacząco maleją po przeprowadzeniu redukcji grafu. Dla przykładu, graf z Rys. 5 definiowany jest przez profile: |V| = 16, |E|=70, d(G)=0,29, indeg(V):1-13, outdeg(V):1-11, a jego zredukowana wersja (istotność powyżej 10% liczby wszystkich zgłoszeń): |V|=8, |E|=10, d(g)=0,09, indeg(V):1-2, outdeg(V):0-2. Więcej przykładów wygenerowanych grafów przedstawiłem w rozdziale 5.2.7. oraz dodatku 8.2. Ponadto wiele innych grafów przedstawiłem w poprzednich pracach [10, 12, 14]

5.2.1. Przepływ zgłoszeń w modelu IHG

Zgłoszenia zgodnie z modelem IHG mogą być przetworzone sekwencyjnie. Przepływ stanu S_j określam jako liczbę zgłoszeń przechodzących przez stan według wzoru:

$$IF(S_j) = \sum_{i \in I} (n_{ij}) + dp(S_j) \qquad (5.5)$$

gdzie J jest zbiorem indeksów określających krawędzie wchodzące do stanu Sj, a n_{ij} określa liczbę zgłoszeń przechodzących przez krawędź e_{ij} ze stanu S_i do S_j. Wprowadzony parametr $dp(S_j)$ określa liczbę zgłoszeń dodanych bezpośrednio przez reporterów w stanie S_j, który można traktować jako stan początkowy. W praktyce wyróżnia się zwykle jeden dominujący stan początkowy. Niemniej jednak, sporadycznie pojawiają się przypadki z dodatkowymi stanami początkowymi. Względną metrykę przepływu (przepustowość – pokrycia zgłoszeń), określa wzór:

$$RIF(S_j) = \frac{IF(S_j)}{N} \qquad (5.6)$$

gdzie: N jest liczbą wszystkich zgłoszeń zarejestrowanych dla projektu.

W praktyce dla niektórych zgłoszeń w ścieżce obsługi często występuje kilkukrotnie ten sam stan tworząc pętle. Opracowałem mechanizmy wykrywania takich pętli w powiązaniu

z przepływem zgłoszeń, uwzględniając liczbę iteracji (sekcja 5.2.5). Ważnym aspektem modelu IHG jest wyznaczenie profili stanów. Spośród nich wyróżniamy 4 podstawowe profile [14]:

1) *sf* – profil przepływu zgłoszeń przez stan projektu P określonego jako zbiór (wektor) parametrów $RIF(S_j)$ ze wszystkimi stanami S_j używanymi w projekcie P
2) *su* – profil wykorzystania stanu będący zbiorem (wektorem) średniej częstotliwości występowania stanów dla ścieżek obsługi zgłoszeń w projekcie P (niektóre stany mogą pojawiać się częściej niż raz w ścieżce z uwagi na pętle) ze wszystkimi *su(Sj)* powiązanymi z podsekwencjami stanu:

$$su(S_j) = \frac{\mathrm{Ir}(S_j)}{N} \qquad (5.7)$$

3) *sr* – profil redundancji stanów lokalnych zdefiniowany jako zbiór (wektor) parametrów we wszystkich stanach w stanie *Sj*:

$$sr(S_j) = \frac{\mathrm{Ir}(S_j) - UIr(S_j)}{\mathrm{Ir}\ (S_j)} \qquad (5.8)$$

4) *sr** – globalny profil redundancji stanu określony jako zbiór (wektor) parametrów we wszystkich stanach w projekcie P:

$$sr^*(S_j) = \frac{\mathrm{Ir}(S_j) - USr(S_j)}{\mathrm{N}} \qquad (5.9)$$

gdzie: $\mathrm{Ir}(S_j)$ określa liczbę wszystkich zgłoszeń obsłużonych w stanie S_j, $UIr(S_j)$ określa liczbę unikalnych zgłoszeń obsłużonych w stanie S_j, N jest liczbą wszystkich zgłoszeń zarejestrowanych dla projektu.

Współczynnik *sr* pokazuje obraz efektu pętli stanów zgłoszeń; $sr(S_j)$>0 określa pętlę zgłoszenia w stanie S_j, w ujęciu lokalnym stanu; parametr *sr** odzwierciedla zbędne przetwarzanie w perspektywie globalnej w odniesieniu do wszystkich zarejestrowanych zgłoszeń. Określam skumulowany przepływ, użytkowanie i redundancję parametrów *AF, AU, AR i AR** odpowiednio jako sumy po wszystkich stanach profili *sf, su, sr* oraz *sr**. Wszystkie te parametry pozwalają na ewaluację procesu obsługi zgłoszeń, w szczególności obciążenia używanych stanów (w odniesieniu do różnorodności ścieżek lub wydajności przepływów i pętli). Przykłady metryk *sf, su, sr* oraz *sr**, przedstawiłem w [14]. Parametry *AF, AU, AR, AR**

zostały wyznaczone również dla realizowanego projektu komercyjnego (innego niż projekt P1) i uwzględniłem je w [11].

5.2.2. Algorytmy analizy grafu IHG

Analiza procesu obsługi zgłoszeń według modeli IHG wspierana jest przez szereg opracowanych algorytmów wśród których wyróżniam 3 grupy: przetwarzanie danych, agregacja danych, algorytmy pomocnicze.

Przetwarzanie danych:

Dane wejściowe programu (informacje o zgłoszeniach) pobierane są z aplikacji JIRA poprzez dostępne serwisy REST API: */search* zwracającego listę zgłoszeń dla określonego zapytania z ograniczeniami (np. data stworzenia zgłoszenia, aktualny status, priorytet bądź typ zgłoszenia), */comment* zwracającego komentarze dla określonego zgłoszenia JIRA. Opracowana aplikacja *IssueAnalyzerTool* pozwala na filtrację weryfikowanych zgłoszeń do konkretnego typu, stanu finalnego, zakresu dat dodania zgłoszeń do repozytorium bądź określonego użytkownika lub listy użytkowników zgodnie z opracowanym modelem IHG. Możliwe jest również pobranie kilku typów zgłoszeń bądź wszystkich zgłoszeń z repozytorium JIRA. Zaimplementowany przeze mnie program ujednolica pobrane dane dotyczące poszczególnych zgłoszeń, łącząc informacje z dwóch serwisów API *(/search* oraz */comment*) w jeden obiekt (zawierający wszystkie pola zgłoszenia JIRA, komentarze i historię zmian zgłoszenia np. zmiana stanu, dodanie komentarza, aktualizacja opisu zgłoszenia, zmiana wartości atrybutu zgłoszenia). Algorytm pobierania danych jest stosunkowo prosty i można zapisać go ogólnymi krokami:

1. Wykonaj zapytanie zgodnie z założeniami wejściowymi (typ zgłoszenia, priorytet, czas dodania zgłoszenia itp.)
2. Dla każdego zgłoszenia z otrzymanej odpowiedzi pobierz listę komentarzy.
3. Przekształć otrzymaną listę do formatu JSON. Wsparcie formatu JSON pozwala w łatwy sposób zapisać zgłoszenia do wewnętrznej bazy danych MongoDB.

Lista pobranych zgłoszeń jest przetwarzana przez algorytm przedstawiony w pseudokodzie Alg. 6. Po sprawdzeniu wielu repozytoriów zaobserwowałem, że lista dotycząca historii zmian zgłoszenia dla projektu X, może być zwrócona w sposób posortowany od najstarszej do najmłodszej zmiany, projekt Y może zwrócić tę samą listę posortowaną w odwrotnej kolejności lub w sposób nieposortowany. Dla uzyskania prawidłowych wyników algorytm dokonuje sortowania listy zmian zgłoszenia względem faktycznego czasu wykonanej

operacji (*linia 10* Alg. 6). Zmiana statusu zgłoszenia wykrywana jest przez porównanie pola, którego dotyczy modyfikacja, z nazwą pola „status” (*linie 23-31* Alg. 6).

```
input: List of JIRA issues with comments and history [I]
output: List of new defined Paths objects [P] which were used during
processing of each jira ticket
```

```
1 function translate_to_path([I]):
2    result = new empty list
3    FOR I in [I] DO
4         P = new empty Path
5         SL = new empty List
6         CNL = new empty list
7         CTL = new empty list
8         CrD = I.createdDate
9         CTL.add(CrD)
10        sort I.history by create date for each item in history
11        FS = Null
12        FOR each item in I.history DO
13            IF (item field equals "status") THEN
14                FS = item.getFromString()
15                BREAK
16            ENDIF
17        IF FS == Null THEN
18            FS = I.currentStatusName()
19        ENDIF
20        CNL.add(firstStatus)
21        FOR each i in I.history DO
22            SN = Null
23            IF (i field == "status") DO
24                IF i.resolution exists THEN
25                     SN = i.fromString + i.resolution
26                ELSE
27                     SN = i.fromString
28             ENDIF
29                CNL.add(statusName)
30                CTL.add(item.getTime)
31            ENDIF
32        ENDFOR
33        FOR i starting from 0 to len(CTL) DO
34            TS = 0
35            IF(i < len(CTL) - 1) THEN
36                TS = CTL[i+1] - CTL[index]
37            ENDIF
38            SL.add(new S(CNL[i], TS))
39        ENDFOR
40        P.SL = SL
41        FOR each C in I.comments DO
42            WC = C.countWords()
43            UWC = C.countUniqueWords
44            P.CL add C.author, C.text
45            P add WC and UWC to proper lists and sort (ascending) them
46        ENDFOR
47        result.add(P)
48    ENDFOR
49    return result
50 endfunction
```

Alg. 6 Pseudokod algorytmu budowania ścieżek z listy zgłoszeń JIRA

Algorytm budowania (Alg. 6) ścieżek wykorzystuje dwie listy pomocnicze, listę nazw kolejno ustawianych statusów (*CNL*) w zgłoszeniu oraz listę czasów zmian statusów (*CTL*). W uproszczeniu algorytm wykonuje następujące kroki dla każdego zgłoszenia z listy wejściowej:

1. Wyznacz pierwszy stan zgłoszenia (*linie 12-19*)
2. Zbuduj listę pomocniczą zmian statusów oraz listę czasów tych zmian (*linie 21-31*)
3. Zbuduj listę stanów ustawianych dla zgłoszenia z czasami ich obsługi na podstawie list pomocniczych (*linie 33-39*) i stwórz nowy obiekt ścieżki
4. Wyznacz statystyki komentarzy i zapisz je do obiektu ścieżki (*linie 41-46*)

Algorytm ustawia czas przebywania zgłoszenia w ostatnim stanie na 0 (*linie 34-38*). Wynikiem algorytmu jest lista ścieżek, w którym jedna ścieżka (warz z parametrami) odpowiada jednemu przetworzonemu zgłoszeniu.

Agregacja danych:

Graf IHG budowany jest przez Alg. 7, który agreguje dane ścieżek wygenerowanych z Alg. 6. Algorytm wyznacza szereg statystyk stanów (liczba wszystkich ustawień statusu w procesowanych ścieżkach, liczba unikalnych ustawień statusu rozumiana jako liczba zgłoszeń, w których wystąpił status, czasy obsługi stanu MIN, AVG, MAX, Kwartyl Q1, Q2, Q3), oraz unikalnych ścieżek (liczba obsłużonych zgłoszeń, statystyki czasowe (MIN, AVG, MAX, Kwartyl Q1, Q2, Q3), statystyki komentarzy (MIN, AVG, MAX), statystyki słowników komentarzy (MIN, AVG, MAX), statystyki unikalnych słów komentarzy (MIN, AVG, MAX), anomalie ścieżek). Statystyki zapisywane są do plików *excel*, które mogą być wykorzystane do szczegółowej (drobnoziarnistej) analizy procesu obsługi zgłoszeń (generowanie plików opisane jest w grupie algorytmów pomocniczych).

Pojedyncza ścieżka (*P*) odpowiada jednemu zgłoszeniu JIRA. Możliwe są wystąpienia ścieżki o tych samych stanach, jednak z różnymi czasami obsługi oraz komentarzami wynikającymi z różnych zgłoszeń. Informacje te są wartościowe do przeprowadzenia bardziej szczegółowych analiz procesu obsługi zgłoszeń. Perspektywa skumulowana wyznaczana jest jako obiekt unikalnej ścieżki (*UP)* reprezentujący grupę zgłoszeń, które przeszły przez tę samą ścieżkę (*linie 39-44* w Alg. 7).

```
input: List of paths ([P]) containing information of statuses set in ticket processing, time
spend in individual states and overall path, comments, ascending sorted list of words count
([WC]) in each comment, ascending sorted list of unique words count ([UWC]) in each comment.
output: Function generates 3 output structures: - Graph (G) containing vertex and edges
representing individual states and paths between them. - file with statistical data of paths -
file with statistical data of states (vertex of graph)
```

```
1 function make_graph([P]):
2    G = new empty graph
3    UPL = new empty List
4    FOR each P in [P] DO
5        US = new empty List
6        FOR each S in P.SL:
7            IF (S not in US) THEN
8                US.add(S.name)
9            ENDIF
10       ENDFOR
11       VL = G.getVerticies()
12       FOR each v in VL DO
13            IF (v.name in US) THEN
14                v.UVC += 1
15            ENDIF
16       ENDFOR
17       FOR each S in P.SL DO
18            IF (G contains V with name equals to S.name) THEN
19                V.count += 1
20                V.TL = S.time
21                IF (S(i+1) and G contains vertex V(i+1) with name equals to S(i+1).name
22                and G contains edge E<V(i),V(i+1)>) THEN
23                    E.count += 1
24                ELSEIF (S(i+1) and G contains vertex V(i+1) with name equals to S(i+1).name
25                and G does not contains edge E<V(i),V(i+1)>) THEN
26                    E = new E<V(i),V(i+1)> and count=1
27                    G.addEdge(E)
28                ELSEIF (S(i+1) and G does not contains vertex V(i+1)) THEN
29                    V = new V<S(i+1).name> and count = 0
30                    G.addVertex(V)
31                    E = new E<V(i),V(i+1)> with count = 1
32                    G.addEdge(E)
33                ENDIF
34            ELSE
35                V = create new V(S.name)
36                V.timeList add(S.time)
37                V.uniqueCount = 1
38                V.count += S.count
39                G.addVertex(V)
40            ENDIF
41        ENDFOR
42        IF (UPL contains P) THEN
43            UPL[P].update(P.statistics)
44        ELSE
45            UP = create new UP(P)
46            UPL.add(UP)
47        ENDIF
48   ENDFOR
49   generatePathsFile(UPL)
50   generateVertexFile(G)
51   paintGraph(G)
52   return G
53 endfunction
```

Alg. 7 Pseudokod algorytmu budowania grafu IHG

Graf G budowany jest na podstawie kolejnych ścieżek (każda ścieżka reprezentuje jedno zgłoszenie) z listy wejściowej algorytmu (Alg. 7 parametr *[P]*). Operacje wykonywane przez algorytm dla każdej ze ścieżek można zapisać w uproszczonej postaci:

1. Dla każdego stanu w ścieżce stwórz listę unikalnych ustawień statusów i jeżeli istnieją w grafie G zaktualizuj odpowiednie wierzchołki (*linie 6-16)*

2. Dla każdego stanu w ścieżce:
    a. sprawdź czy istnieje wierzchołek w grafie z taką samą nazwą jeżeli tak zaktualizuj wierzchołek (19-20), jeżeli nie dodaj nowy wierzchołek (*linie 35-39*)
    b. Sprawdź czy istnieje krawędź łącząca $stan_i$ ze $stanem_{i+1}$ jeżeli tak zaktualizuj krawędź (*linie 21-23*). jeżeli nie dodaj ją (*linie 24-32*)
3. Jeżeli ścieżka istnieje w liście ścieżek unikalnych zaktualizuj ją, jeżeli nie dodaj ścieżkę do listy ścieżek unikalnych (*linie 42-47*)

Stany występujące w ścieżce mapowane są na wierzchołki grafu (*V*), które definiowane są na podstawie nazw stanów ścieżki *(linia 35* Alg. 7)). Każdy z wierzchołków zawiera dwa liczniki, licznik liczby ustawień statusu we wszystkich zgłoszeniach (*count – linia 38* Alg. 7)) oraz licznik unikalnych wystąpień statusu (*linia 37* Alg. 7), tożsamego z liczbą ścieżek, w których występuje. Skierowane krawędzie grafu (E) definiują przejścia pomiędzy stanami, algorytm buduje krawędzie na podstawie następujących po sobie zmianach statusu ($status_i$, $status_{i+1}$ *– linie 21-32* Alg. 7)). Elementem wyjściowym algorytmu jest graf G<V,E> zgodny z opracowaną koncepcją IHG, w którym V reprezentuje zbiór wierzchołków grafu, a zbiór E listę skierowanych krawędzi grafu.

Algorytmy pomocnicze

Jedną z funkcji programu (*IssueAnalyzerTool*) jest wizualizacja grafu IHG generowanego przez Alg. 7., którą reprezentuje funkcja pomocnicza *paintGraph (linia 51* Alg. 7*)*. Do rysowania grafu wykorzystuję bibliotekę *JUNG* [106], która wspiera wizualizację grafu według współrzędnych wyznaczanych przez algorytm, które kroki można zapisać następująco:

1. Wyznacz stany początkowe grafu G
2. Podziel wierzchołki grafu G na grupy: *pierwsza grupa* – wierzchołki stanów początkowych grafu (G), *druga grupa* – wierzchołki połączone z wierzchołkami *grupy pierwszej* krawędziami wychodzącymi (wierzchołki następujące po grupie pierwszej), *trzecia grupa* – wierzchołki połączone z wierzchołkami *grupy drugiej* krawędziami wychodzącymi (wierzchołki następujące po *grupie drugiej*) … aż do pokrycia wszystkich wierzchołków
3. Dla każdej z grup ustaw współrzędne wierzchołka grafu według wzorów: *x=100*nr_grupy; y= 100*nr_wierzchołka_w_grupie*
4. Zaprezentuj graf

Program *IssueAnalyzerTool* pozwala również użytkownikowi na ręczną modyfikację grafu poprzez przesuwanie jego wierzchołków w strefie roboczej okna aplikacji.

Dostępna jest również funkcja filtracji wygenerowanego grafu względem istotności krawędzi bądź wierzchołków. Algorytmy redukcji grafu IHG można zapisać w postaci dwóch funkcji, *filter_vertices* (Alg. 8) dla filtracji wierzchołków oraz *filter_edges* (Alg. 9) dla filtracji krawędzi.

```
input: integer value of percent number (FP) for which Graph vertices will be
filtered out. Algorithm consider max percent value as a highest count value
of vertices in graph
output: filtered graph (vertex with count lower than input percent are
hidden)
```

```
1 function filter_vertices(FP):
2    MAXVC = 0
3    VL = G.getVerticies()
4    FOR each V in VL DO
5        IF (V.C > MAXVC) THEN
6            MAXVC = V.C
7        ENDIF
8    ENDFOR
9    FC = MAXVC * FP / 100
10   FOR each V in VL DO
11       IF V.count < FC THEN
12           V.setVisibility(FALSE)
13       ELSE
14           V.setVisibility(TRUE)
15       ENDIF
16    ENDFOR
17 endfunction
```

Alg. 8 Pseudokod algorytmu filtracji wierzchołków

```
input: integer value of percent number (FP) for which Graph (G) edges will be
filtered out. Algorithm consider max percent value as a highest count value
of edges in graph
output: filtered graph (edges with count lower than input percent are hidden)
```

```
1 function filter_edges(FP):
2    MCE = 0
3    EL = G.getEdges()
4    FOR each E in VL DO
5        IF E.count > MCE THEN
6           MCE = E.count
7        ENDIF
8    ENDFOR
9    FC = MCE * FP / 100
10       FOR each E in EL DO
11          IF E.count < FC THEN
12             E.setVisibility(FALSE)
13          ELSE
14             E.setVisibility(TRUE)
15          ENDIF
16       ENDFOR
17 endfunction
```

Alg. 9 Pseudokod algorytmu filtracji krawędzi

W obydwu przypadkach algorytmy filtracja (Alg. 8, Alg. 9) odwołują się do maksymalnej liczby przejść przez stan (dla wierzchołków) lub maksymalnej liczby przejść przez krawędź (dla krawędzi) jako maksymalnej wartości funkcji (*linie 5-7* Alg. 8, Alg. 9). Jeżeli procent wartości pokrycia wierzchołka (stanu) bądź krawędzi jest mniejszy od wskazanej liczby progowej, element jest ukrywany *(linia 11-12* Alg. 8, Alg. 9). Funkcjonalność filtracji pozwala w łatwy sposób zaprezentować na grafie jedynie interesujące użytkownika ścieżki bez ścieżek i wierzchołków o znikomej istotności. Taki szum informacyjny jest bardzo problematyczny w przypadku generowania grafów dla dużej liczby zgłoszeń z nieuporządkowanym procesem obsługi. Pozwala to także na jednoznaczne określenie zbędnych stanów procesu obsługi zgłoszeń.

Opracowany model IHG wyznacza szereg profili ścieżek i krawędzi grafu które zapisywane są do dedykowanych plików CSV (linie 49 – 50, Alg. 7). Pseudokod generowania plików przedstawiłem w Alg. 10 dla stanów grafu oraz w Alg. 11 dla unikalnych ścieżek grafu. Wyjściowe pliki CSV, dzięki generycznej postaci mogą być łatwo wykorzystane przez użytkownika w kolejnych analizach np. przy użyciu programu *Microsoft Excel*.

```
input: graph (G) with <Vertex, Edges>
output: CSV file (F) which can be later used in excel, with statistics of
state usage in graph G including data:
    - time statistics like MIN, AVG, MAX, Q1, Q2, Q3
    - count of unique states appearance in paths
    - count of state appearance in paths
```

```
1  function generate_states_file(G)
2    VL = G.getVertices()
3    F = create new empty CSV file with headers
4    FOR each V in VL DO
5        status = V.name
6        UC = Vertex.uniqueCount
7        C = Vertex.count
8        minT, avgT, maxT, Q1, Q2, Q3 = calculateTimeStatistics(V.timeList)
9        Sf, Sr*, Sr, Su = calculateStateStatistics()
10       L = new empty list
         L.add(V.name, UC, C, minT, avgT, maxT, Q1, Q2, Q3, Sf, Sr*, Sr, Su)
11       F.addNewLine(listOfValues)
12   ENDFOR
13   return F
14 endfunction
```

Alg. 10 Pseudokod algorytmu generowania pliku stanów

Parametrem wejściowym Alg. 10 jest finalny graf IHG (*G*), z którego pobierane są wszystkie wierzchołki (*linia 2*) wraz z listami czasów i licznikami wystąpień statusów w ścieżkach,

na podstawie których generowane są statystyki czasowe (MIN, AVG, MAX, Kwartyl Q1, Q2, Q3) w linii 8 oraz charakterystyki profili stanu (*Sf*, *Sr**, *Sr*, *Su*) w *linii 9*. Informacje dotyczące

każdego z wierzchołków grafu zapisywane są w pliku wyjściowym Alg. 10 (*F*) jako pojedyncze wiersze (*linia 11*)

```
input: list of UniquePath ([UP]) which accumulates information of all paths
with the same flow
output: CSV file (F) with statistics of unique paths:
    - time statistics like MIN, AVG, MAX, Q1, Q2, Q3
    - comments statistics like comments count(C MIN, AVG, MAX),
      comments words count (min, avg, max), comments unique words count(MIN,
      AVG, MAX)
    - loops found within a path
    - count of issues which was handed by path
```

```
1  function generate_paths_file([UP])
2    F = create new empty CSV file with headers
3    FOR each UP from [UP] DO
4        C = UP.count
5        SL = new empty List
6        FOR each S in UP.path DO
7              SL.add(S.name)
8        ENDFOR
9        L = find_loops(UP.path)
10       minT, avgT, maxT, Q1, Q2, Q3 = calculateTimeStatistics(UP.timeList)
11       minCC, avgCC, maxCC = caluculateCountStatistics(UP.comments)
12       minCWC, avgCWC, maxCWC = caluculatCountStatistics(UP.CWC)
13       minCUWC, avgCUWC, maxCUWC = calculateCountStatistics(UP.CWUC)
14       TMPL = new empty List
15       TMPL.add (C, SL, L,
16                 minT, avgT, maxT,
17                 Q1, Q2, Q3,
18                 minCC, avgCC, maxCC,
19                 minCUWC, avgCUWC, maxCUWC)
20       F.addNewLine(listOfValues)
21   ENDFOR
22   return F
23 endfunction
```

Alg. 11 Pseudokod algorytmu generowania pliku unikalnych ścieżek

Alg. 11 przedstawia pseudokod algorytmu generowania pliku ze statystykami unikalnych ścieżek grafu. Każda z nich opisana jest przez: stany ścieżki, liczbę zgłoszeń, w których wystąpiła wskazana ścieżka oraz statystyki czasowe (MIN, AVG, MAX, kwartyl Q1, Q2, Q3), dane statystyczne komentarzy występujących w poszczególnych zgłoszeniach takie jak min, avg, max liczby komentarzy, liczby wyrazów w komentarzach oraz liczby unikalnych wyrazów w komentarzach. Plik generowany jest na podstawie listy unikalnych ścieżek przyjmowanych na wejściu Alg. 11. Do kalkulacji statystyk czasowych wykorzystywana jest funkcja *calculateTimeStatistics()*, bazująca na listach czasów obsługi ścieżki (*linia 10*). Funkcja *calculateCountStatistics()* zlicza wartości min, avg, max na podstawie list liczby komentarzy (*linia 11*), słów komentarzy (*linia 12*) oraz unikalnych słów komentarzy (*linia 13*). Dane dają pewien obraz wielkości słowników oraz częstości interakcji aktorów w procesie obsługi

zależnie od ścieżki jaką przeszło zgłoszenie. Charakterystyki unikalnych ścieżek zapisywane są jako pojedyncze wiersze w generowanymi pliku (*F*) Alg. 11 (*linia 20*)

Wartościowym aspektem zaimplementowanego programu (*IssueAnalyzerTool*) jest wyznaczanie anomalii (pętli) ścieżek. Są one uwzględniane w pliku wyjściowym Alg. 11. Algorytm wyznaczania anomalii ścieżek sprawdza zapętlenia następujących po sobie statusów. W przypadku wystąpienia pętli tworzona jest nowa anomalia określona jako zapętlony ciąg stanów, rząd anomalii definiowany poprzez liczbę stanów w pętli. Rząd 1 anomalii oznacza że w procesie obsługi zgłoszenia określony status był ustawiony dwukrotnie (lub więcej razy) bez stanu pośredniego. Świadczy to bezpośrednio o błędzie w systemie lub niespójności danych w repozytorium. Opracowany przeze mnie algorytm którego pseudokod przedstawia Alg. 12, zakłada poszukiwanie anomalii polegających na pełnym powtórzeniu wszystkich stanów podciągu bez przerwań tzn. stan końcowy pętli jest jednocześnie stanem początkowym powtarzającej się pętli.

```
input: Path (P) represented as a list of statuses for given issue, list
should be sorted in chronological order based on status change date which is
a part of earlier algorithm
output: List of found loops ([AL]) defined as anomaly (A)  with two
parameters, looped state or states, severity of a loop - 1 is the most
critical which means the same state is appearing after itself in path
```

```
1 function find_loop(P):
2    AL = new empty List
3    S = P.getStatuses()
4    FOR LL = 1 to (length of list)/2 DO
5        IF (LL == 1) THEN
6            FOR I=1 to len(S)-1 DO
7                IF (S[I] == S[index+1] THEN
8                     AL.add(new A(list[i], 1))
9                ENDIF
10           ENDFOR
11       ELSE:
12            I = 0
13            WHILE (I + 2 * LL < len(S)) DO
14                L = new list as sublist of S with beginning in
                      S[I] and end in S[I + LL]
15                LTC = new list as sublist of S with beginning in
                        S[I + LL] and end in list[(I+2)*(LL-1)]
16                IF (L equals LTC) THEN
17                    AL.add(new A(L, LL))
18                ENDIF
19                I +=1
20            ENDWHILE
21        ENDIF
22    ENDFOR
23    return AL
24 endfunction
```

Alg. 12 Pseudokod algorytmu wyszukiwania pętli

Dla zobrazowania, dla ścieżki statusów: [*O,O,R_F,C,RE,R_F,C,RE,R_F,C*] algorytm wyznaczy dwie anomalie: *{[O:1]; [C;RE;R_F;C:4]}*. Oznacza to, że stan *O* pojawia się dokładnie po samym sobie, podobnie jak pełna sekwencja stanów *C;RE;R_F;C* która jest powtarzana zakładając, że kolejna pętla zaczyna się od stanu finalnego ciągu tzn *C*. Algorytm przedstawiony został w formie pseudokodu (Alg. 12) funkcji *find_loop*.

Zaprezentowane algorytmy wspierają generowanie szeregu charakterystyk ścieżek oraz stanów grafu, które mogą zostać zagregowane w następujące grupy:

- Statystyki ścieżek:
  - Czasowe (min, avg, max, kwartyl Q1, Q2, Q3)
  - Strukturalne (stany ścieżki, strukturę pętli stanów w ścieżce, ścieżki kompatybilne)
  - Ilościowe:
    - liczba zgłoszeń przechodzących przez ścieżkę
    - długość ścieżki
    - liczba pętli
  - Komentarzy zgłoszeń przechodzących przez ścieżkę
    - Ilościowe (min, avg, max liczby komentarzy)
    - Słownikowe: minimalną, średnią, maksymalną liczby:
      - Słów w komentarzach
      - Unikalnych słów w komentarzach
- Statystyki stanów:
  - Czasowe (min, avg, max, kwartyl Q1, Q2, Q3)
  - Ilościowe (liczbę ustawień stanu, liczba unikalnyc ustawień stanu)
  - Strukturalne (profile *Sf, Sr*, Sr, Su)*

które generowane są w postaci plików CSV możliwych do szczegółowej analizy w programie *Excel*. Strukturę pliku przedstawiłem w załączniku 8.3, a jego przykład w załączniku 8.4.

Wszystkie opracowane algorytmy opisane w sekcji 5.2.2. wspierają model IHG. Oznacza to że, mogą generować wyniki dla grafu pełnego lub z ograniczeniami R, np. dla określonego typu zgłoszenia, stanu, określonej grupy reporterów tworzących zgłoszenia lub czasu, w którym zostały one dodane do repozytorium. Więcej analiz ścieżek i stanów zgłoszeń przedstawiłem w rozdziałach 5.2.3, 5.2.4, 5.2.6.

5.2.3. Charakterystyki czasowe stanów grafu IHG

Zgłoszenia rozpatrywane w ramach modelu IHG procesowane są przez aktorów poprzez zmianę stanu zgłoszenia ze stanu S_i do stanu S_{i+1} zależnie od wykonanej przez nich akcji. Przykładowo, zgłoszenie po dostarczeniu poprawki przejdzie ze stanu *In Progress* do *Resolved_Fixed*. Śledząc statystyki czasowe obsługi zgłoszeń w modelu IHG możemy określić czasy obsługi poszczególnych stanów poprzez profile MIN, AVG, MAX, Kwartyl Q1,Q2,Q3 czasu przebywania zgłoszenia w stanie. Określając te profile, możemy zdefiniować ograniczenia badanych zgłoszeń, takie jak: typy zgłoszeń (*Bug, New Feature, Task, User Story, Improvement*, lub wszystkie typy), priorytety zgłoszeń, reporterów tworzących zgłoszenia, czas w jakich zgłoszenia zostały stworzone lub rozwiązane, aktualny status zgłoszenia (zgłoszenie w trakcie procesowania lub zamknięte). Zasadne jest korelowanie tych profili z unikalną liczbą wystąpień stanu (liczba ścieżek, w których występuje stan bez powtórzeń – kolumna U) oraz całkowitej liczby wystąpień stanu w ścieżkach zgłoszeń (włączając powtórzenia stanu - kolumna N).

Stan	U	N	AVG	MAX	Q2	Q3
NES	12644 47	14725 51	1 0,1	799,7 14	0,2 <1h	4,9 0,7
NEM	1728 13	1765 13	0,8 1,7	178,8 6	0,1 0,6	0,7 0,7
INV	1678 6	2430 6	22 31,1	823,9 123,5	1,1 0,3	9 3,1
IPR	8489 44	9702 47	0,4 5,1	373,2 30,4	0,2 0,1	2,8 0,8
OPE	11325 46	13573 48	0,3 0,1	850,9 33,2	1,1 0,1	23,4 0,9
ICR	8767 46	9316 48	3,4 0,7	362,1 31,2	1,9 1	5,8 3,7
NED	12400 51	12466 51	0,2 0,1	322,9 6,9	<1h <1h	<1h 0,2
C	11811 54	12575 55	40,2 0,2	876,2 0,2	1 0,2	7,3 0,2
NET	88	90	0,1	228,5	0,2	2,8
WFU	437	723	101	481,1	1,9	17,9
BAC	2034 2	2124 2	116,3 0	726 0	17,1 0	70,2 0
BLO	298	332	4,5	814,3	7,1	28,7

Tab. 20 Statystyki czasowe stanów obsługi zgłoszeń projektu MongoDB (13435 zgłoszeń: 8405 błędów, 5030 nowych funkcjonalności).

Dla ilustracji w Tab. 20 przedstawiłem przykład profili stanów zgłoszeń projektu MongoDB uwzględniającego łącznie 13435 zgłoszeń, w tym 8405 zgłoszeń błędów (7895 zamkniętych i 510 otwartych) oraz 5030 zgłoszeń nowych funkcjonalności (*Improvement* (3155 otwartych, 1102 zamkniętych), *New Feature* (625 zamkniętych,

148 otwartych)). Wartości przedstawione w drugim wierszu w komórkach odnoszą się do zamkniętych zgłoszeń błędów o najwyższym priorytecie (54 zgłoszenia).

Zaprezentowane czasy AVG, MAX, Q2, Q3 określone są w dniach. Sumaryczna liczba stanów dla modelu IHG wynosi 22. Dla ułatwienia prezentacji tabela została ograniczona i nie zawiera stanów o małej istotności, przez które przeszły maksymalnie 3 zgłoszenia. Stany nieuwzględnione w tabeli: *REQ(1;1), WAI(3;4), WAR(2;3), CLO(2;2), DEU(3;4), WAC(2;3), SCH(2;2), DWS(1;2), RES(1;3),* w których pierwsza liczba reprezentuje liczbę zgłoszeń ze stanem, druga określa liczbę ustawień stanu. Nazwy stanów zaprezentowane są w skróconej formie zgodnie z przyjętym nazewnictwem: *Requested – REQ, Wait - WAI, Needs Scheduling – NES, Needs Merge – NEM, Waiting for Reporter – WAR, CLOSED – CLO, Investigation – INV, Debugging – DEU, In Progress – IPR, Waiting for Customer – WAC, Open – OPE, In Code Review – ICR, Scheduled – SCH, Needs Verification – NED, Closed – Clo, Needs Triage – NET, Waiting For User Input – WFU, Backlog – BAC, Debbuging with Submitter – DWS, Blocked – BLO, Researching – RES.*

Profile stanów z uwagi na niedoskonałości w procesowaniu zgłoszeń obarczone są możliwym błędem zaniżenia czasu (np. brak ciągłości w procesowaniu zgłoszeń zaobserwowany w projekcie Log4J2, którego statystyki czasów dostępne są w załączniku 8.3. Algorytm (Alg. 7) ustawia czas przebywania w stanie na 0, w przypadku kiedy stan jest ostatnim stanem w ścieżce zgłoszenia. W konsekwencji, jeżeli dla pewnej liczby zgłoszeń stan był pośredni a dla innej grupy zgłoszeń był ostatnim ustawionym stanem czas jego obsługi może być zaniżony.

Warto zaznaczyć, że wysokie czasy obsługi mogą być związane ze zgłoszeniami, które zostały odroczone na późniejszy moment lub ich procesowanie nie jest kontynuowane. Sytuacja ta mogłaby mieć znaczny wpływ na zwiększenie wartości parametrów Q2 oraz Q3. Projekt MongoDB nie wykazuje takiej tendencji. Profile czasowe mogą być wygenerowane dla wszystkich zgłoszeń bądź oddzielnie dla zgłoszeń zamkniętych i otwartych. W praktyce dane nie różnią się istotnie, normalną sytuacją jest mniejsza liczba zgłoszeń otwartych niż zamkniętych. Model IHG zakład również możliwość ograniczenia priorytetów weryfikowanych zgłoszeń, po przeanalizowaniu wielu repozytoriów widoczna jest także tendencja do obsługi zgłoszeń z najwyższym priorytetem w o wiele krótszym czasie niż reszty zgłoszeń. Maksymalny czas obsługi może być nawet 100 razy większy od czasu Q3. Analiza czasów obsługi stanów modelu IHG pozwala wykryć pewne anomalii procesu obsługi zgłoszeń

(np. szczególnie problematycznych stanów, w których zgłoszenia przebywają za długo) lub określić jednostkowe przypadki wpływające na czas obsługi stanu.

Analiza wielu projektów ujawniła także różnice pomiędzy nimi. Istotną różnicą są obsługiwane stany, dla projektu MongoDB większość występujących stanów jest skonfigurowana w dedykowany sposób odpowiedni dla projektu, jednak utrudnia analizy porównawcze z innymi projektami. Dla porównania w projekcie Groovy możemy zidentyfikować 20 stanów, z których 15 stanowią wariant stanu *Resolved_X*. Projekt MongoDB ma o wiele bardziej skomplikowany proces przepływu zgłoszeń. Nadmierna komplikacja procesu może także wpływać na czas jego obsługi.

Czasy występujące w projekcie MongoDB są najniższe, w porównaniu z innymi badanymi projektami *open source*. Może to być związane z wysoką popularnością projektu w ostatnich latach oraz świadczyć o dobrym sposobie organizacji prac. Dla porównania w projekcie Groovy czas Q3 dla stanu *Open* (OPE) wynosił 153 dni dla zamkniętych zgłoszeń błędów, w MongoDB wartość ta równa jest 8 dni. Zazwyczaj profile czasowe projektów *open source* są gorsze od projektów komercyjnych, gdzie spora uwaga przywiązywana jest do prawidłowego procesowania zgłoszeń. W projekcie komercyjnym P1 (technologia *SCRUM*) czas Q3 dla stanu początkowego New zamkniętych zgłoszeń błędów wynosił 11,9 dnia, co jest lepszym czasem od projektu Groovy dla którego miara ta jest równa 153 dni (stan początkowy *Open*). Profile czasowe projektu P1 dla pełnego grafu IHG (z uwzględnieniem zgłoszeń błędów i nowych funkcjonalności) są wydłużone przez zgłoszenia nowych funkcjonalności, których procesowanie jest długie. Istotne jest jednak, aby ich kwartyl Q3, był niższy niż czas 1 sprintu. Wyższy wartości świadczyłby o niepożądanej sytuacji niedostarczenia zgłoszenia w zaplanowanym sprincie (tzw. *spillover*). Perspektywa czasowej obsługi zgłoszeń jest omówiona dokładniej w sekcji 5.2.6., załączniki 8.4. zawiera opis i wycinek pliku *excel* zawierającego szerokie statystyki badanych projektów.

5.2.4. Profile ścieżek obsługi zgłoszeń w modelu IHG

Perspektywa metryk zaprezentowana w sekcji 5.2.3, dotyczy stanów obsługi zgłoszeń. Innym ujęciem procesu obsługi zgłoszeń są ścieżki obsługi, tzn. następujące po sobie stany. Możemy tutaj śledzić przepływ zgłoszeń przez daną ścieżkę, strukturę ścieżki, statystyki czasowe przejść przez ścieżkę (MIN, AVG, MAX, kwartyl Q1,Q2,Q3) oraz anomalie występującym w ścieżkach (np. pętle stanów). Profile ścieżek zgodnie z koncepcją IHG mogą dotyczyć pełnego grafu (wszystkie typy zgłoszeń) lub z ograniczeniem R

(np. na: typ zgłoszenia, priorytet, reporter, stan finalny, przedział czasowy stworzenia zgłoszenia itp.) dla zawężenia prowadzonej analizy.

Wyróżniam dwie klasy profili ścieżek:

- wydajnościowo-czasowe – których analizy przeprowadzane są na podstawie generowanych plików *excel* (wynik działania Alg. 11), umożliwiających sortowanie względem kolumn zależnie od celu analizy np. liczby zgłoszeń obsługiwanych przez ścieżkę lub średniego czasu obsługi, długości ścieżki itp. Tab. 21 przedstawia przykładowy fragment generowanego pliku dla grafu IHG projektu P1 z ograniczeniem R – zgłoszenia błędów posortowanych malejąco względem liczby obsłużonych zgłoszeń przez ścieżki (kolumna N). Pełen widok pliku wyjściowego pliku Alg. 11 zaprezentowałem w załączniku 8.3. Szczegółowe analizy profili wydajnościowo - czasowych prezentuję w sekcji 5.2.6

N	Ścieżka	L	AVG	MAX	Q2	Q3
576	N,IPrS,REV,TbT,V/C	5	25,2	340,2	6,9	21,2
106	N,TbT,V/C	3	29,8	332,3	3,8	17,3
100	N,V/C	2	26,4	218,2	5	17
48	N,IPrS,REV,TbT,Reo,IPrS,REV,TbT,V/C	9	60,3	283,7	30,3	65,4
45	N,NInf,IPrS,REV,TbT,V/C	6	46,1	212,9	15	49,9
23	N,IPrS,REV,TbT,Reo,TbT,V/C	7	30,6	112,1	22,1	36,2
22	N,IPrS,N,IPrS,REV,TbT,V/C	7	45,5	294,2	25,6	74,7
21	N,IPrS,V/C	3	30,3	131,3	10,2	41,3
19	N,NInf,IPrS,V/C	4	58,4	235,4	28,3	80
18	N,IPrS,NInf,IPrS,REV,TbT,V/C	7	48,2	248,7	27,4	49,3

Tab. 21 Wybrane ścieżki z największym pokryciem zgłoszeń projektu P1 (IHG z ograniczeniem R - zgłoszenia błędów, łączna liczba zgłoszeń 1555)

- strukturalne – które obejmują zdefiniowane profile długości ścieżek (PL_P) oraz stanów ścieżek (PS_P) dla projektu P:

$$PL_P = [l1(c1, n1); l2(c2, n2); \dots lk(ck, nk)] \quad (5.10)$$

$$PS_P = \{P_i : S1_i, S2_i, \dots Sm_i | cp_i, 0 \leq i \leq r\} \quad (5.11)$$

Gdzie l1,l2,…,lk; n1,n2,..,nk i c1,c2,..,ck oznaczają odpowiednio długość ścieżek (liczba następujących po sobie stanów) w szeregu rosnącym, liczbę różnych ścieżek o określonej długości oraz sumę zgłoszeń przechodzących przez ścieżki o tej długości. PS_P oznacza zbiór wszystkich ścieżek; P_i składa się z sekwencji stanów $S1_i$,$S2_i$….Sm_i; cp_i określa liczbę zgłoszeń przechodzących przez ścieżkę i. Pokrycie ścieżki może być określone na dwa

sposoby: 1) bezwzględny jako liczbę zgłoszeń, 2) względny np. jako procent liczby zgłoszeń przechodzących przez ścieżkę i względem wszystkich zgłoszeń w grafie. Przykład profilu stanów PS_P dla projektu komercyjnego P1 podano w Tab. 21. (profil cp_i jest oznaczony jako N) Profil dla innych projektów zaprezentowałem w załącznikach 8.1., 8.4.

Model IHG umożliwia wyprowadzenie innych zagregowanych profili np. warianty ścieżek np. ze wspólnym stanem początkowym, wspólnym stanem początkowym i pośrednim, wspólnym stanem początkowym i końcowym itp. W profilach grupujących możemy określić profile liczby ścieżek we wspomnianych perspektywach, względnej i bezwzględnej. Analogicznie profile grupowe mogą zostać zredukowane ograniczeniem R określającym np.: reporterów, typ zgłoszenia, czas stworzenia zgłoszenia lub jego rozwiązania. Przy generacji profili grupowych możemy wykorzystać również wyrażenia regularne określające sekwencję stanów przez jakie przechodziły zgłoszenia.

Wygenerowane profile długości ścieżek (PL_P) dla projektu P1 oraz MongoDB z różnymi ograniczeniami R dotyczącymi typów zgłoszeń oraz priorytetów przedstawiłem poniżej:

Projekt P1

W Projekcie P1 w ciągu 1 roku kalendarzowego raportowano 2378 zgłoszeń, w tym 1555 zgłoszeń błędów (1434 zamknięte i 121 otwartych), oraz 823 zgłoszenia nowych funkcjonalności. W zgłoszeniach nowych funkcjonalności wyróżniamy: 247 zgłoszeń typu *User Story* (209 zamkniętych, 38 otwartych), 342 zgłoszenia typu *Task* (282 zamkniętych, 60 otwartych), 234 zgłoszenia typu *New Feature* (134 zamknięte, 100 otwartych).

Profil PL_P wzbogacony o procent pokrycia ścieżek przez zgłoszenia względem całkowitej liczby zgłoszeń (po myślniku) dla projektu komercyjnego P1:
23(4;4) - 0.17%; 21(4;3) - 0.17%; 20(5;5) - 0.21%; 19(9;9) - 0.38%; 18(8;8) - 0.34%; 17(13;13) - 0.55%; 16(12;12) - 0.5%; 15(18;15) - 0.76%; 14(17;17) - 0.71%; 13(33;24) - 1.39%; 12(16;14) - 0.67%; 11(53;37) - 2.23%; 10(49;27) - 2.06%; 9(138;45) - 5.8%; 8(82;35) - 3.45%; 7(160;43) - 6.73%; 6(341;27) - 14.34%; 5(750;29) - 31.54%; 4(164;16) - 6.9%; 3(214;20) - 9.0%; 2(148;8) - 6.22%; 1(140;2) - 5.89%;

W przypadku modelu IHG z ograniczeniem R zgłoszenia błędu (Bug) uwzględniamy 1555 zgłoszeń, które tworzą 332 unikalne ścieżki. Ścieżki o długościach z zakresu (12-24) obsłużyły 8,54% zgłoszeń, (8-11) 16,01%, (6-7) 14,02%, (5)41,29%, (1-4) 20,13%. Tab. 21 prezentuje 10 ścieżek z największym pokryciem dla wygenerowanego projektu IHG.

Dla zgłoszeń błędów o najwyższym priorytecie (32 zamknięte zgłoszenia) wykryłem 16 unikalnych ścieżek, dla których w zależności od zakresu długości obsłużono (7-20) 21,86%, (5-6) 40,62%, (2-3)37,5%. Warto zaznaczyć, że zgłoszeń o najwyższym priorytecie jest najmniej w porównaniu z innymi priorytetami. Niezależnie od ograniczenia grafu widoczna jest dominacja ścieżek w zakresie długości (5-6).

Projekt MongoDB

Dla projektu MongoDB w okresie 2 lat kalendarzowych wykryłem 13435 zgłoszeń, w których skład wchodziło 8405 zgłoszeń typu *Bug* (7895 zamkniętych, 510 otwartych) oraz 5030 zgłoszeń nowych funkcjonalności uwzględniających 4257 zgłoszeń typu *Improvement* (3155 zamkniętych, 1102 otwartych) oraz 773 zgłoszenia typu *New Feature* (625 zamkniętych, 148 otwartych).

Profil PL_P wzbogacony o procent pokrycia ścieżek przez zgłoszenia względem całkowitej liczby zgłoszeń (po myślniku) dla projektu MongDB:
28(1;1) - 0.01%; 27(1;1) - 0.01%; 23(1;1) - 0.01%; 22(2;2) - 0.01%; 21(8;8) - 0.06%; 20(6;6) - 0.04%; 19(7;7) - 0.05%; 18(9;9) - 0.07%; 17(12;12) - 0.09%; 16(17;17) - 0.13%; 15(26;24) - 0.19%; 14(34;30) - 0.25%; 13(43;39) - 0.32%; 12(102;84) - 0.76%; 11(121;91) - 0.9%; 10(295;158) - 2.2%; 9(400;172) - 2.98%; 8(686;157) - 5.11%; 7(1948;159) - 14.5%; 6(5023;119) - 37.39%; 5(1630;71) - 12.13%; 4(1617;42) - 12.04%; 3(1161;22) - 8.64%; 2(283;7) - 2.11%; 1(2;2) - 0.01%;

W przypadku ograniczonego modelu IHG z R – typ zgłoszenia *Bug* (8405 zgłoszeń błędów - 7895 zamkniętych i 510 otwartych), algorytm wykrył 832 unikalne ścieżki, których zakres pokrywał odpowiednio (10-28) 2,45%, (7-9)21,98%, (6) 41,22%, (2-5) 31,97% procent zgłoszeń.

Wyznaczając model IHG dla projektu MongoDB z ograniczeniem R (zamknięte zgłoszenia typu Bug o najwyższym priorytecie – *Blocker*), wykryłem 15 unikalnych ścieżek, które obsłużyły 54 zgłoszenia. Ścieżki w zakresach długości: (8-10), (6-7), (2-5) obsłużyły odpowiednio 9,26%,72,22%,18,52% zgłoszeń. Zgłoszenia o niższych priorytetach były obsługiwane przez ścieżki z zakresów długości: *Critical* (2-16) 123 zgłoszeń, *Major* (2-28) 7488 zgłoszeń, *Minor/Trivial* (2-14) 231 zgłoszeń. Analizując wiele różnych repozytoriów zgłoszeń *open source* i komercyjnych, zauważyłem, że w każdym projekcie najdłuższe ścieżki najczęściej występują dla domyślnego priorytetu zgłoszeń (*Major, Medium*), maksymalna długość w zależności od projektu występowała z fluktuacją (9-28). Długie ścieżki są różnorodne jednak obsługują małą liczbę zgłoszeń najczęściej po 1 zgłoszeniu. Informacja ta

może stanowić pewien wskaźnik anomalii obsługi zgłoszeń. Stosunek liczby ścieżek obsługujących 1 zgłoszenie względem liczby wszystkich unikalnych ścieżek dla kolejnych priorytetów projektu MongoDB wynosi: *Blocker* (0,66), *Critical* (0,84), *Major*(0,68), *Minor/Trivial* (0,66). Współczynnik dla kolejnych priorytetów zamkniętych zgłoszeń błędów projektu P1 wynosił: *Blocker* (0,81), *Critical* (0,77), *Major* (0,69), *Minor* (0,68), *Trivial* (0,86).

Dla kilku weryfikowanych projektów *open source* otrzymałem typowe profile *(PL_p)* ze ścieżkami w zakresie 2-10 stanów, badane projekty komercyjne posiadały ścieżki w zakresie 1-15 stanów. Niemniej występują również inne długoterminowe projekty komercyjne (np. [14]), których długość ścieżek waha się w zakresie (4-44) zaś liczba ścieżek wynosiła ponad 1500, które pokrywały zgłoszenia w rozkładzie od ułamka procenta do 10%. Było to projekt długoterminowy z dużą fluktuacją aktorów. Co więcej, duża liczba dostępnych stanów tworzy możliwość dużej różnorodności ścieżek obsługi. Większa różnorodność ścieżek jest bardziej prawdopodobna w projektach o dużej liczbie stanów. Pokrycie długiej ścieżki jest zazwyczaj niższe niż typowych ścieżek. Zjawisko to może być śledzone w profilach $PS_{p.}$ Profile $PL_{p\,i}\,PS_p$ ścieżek prezentują sekwencje typowych, optymalnych oraz wyjątkowych ścieżek obsługi zgłoszeń, pokazując perspektywę skumulowaną oraz szczegółową procesu obsługi zgłoszeń modelu IHG. W większości projektów zaobserwowałem grupę 20-30 unikalnych ścieżek, które obsługują ponad 60% zgłoszeń. Z drugiej strony istnieje wiele ścieżek pokrywających pojedyncze liczby zgłoszeń 1-10. Obserwacja dominującej grupy ścieżek jest dobrą oznaką, natomiast duża liczba ścieżek pokrywających pojedyncze zgłoszenia może być uznana za pewną niedoskonałość, która powinna zostać zweryfikowana ręcznie i przedyskutowana z aktorami projektu.

5.2.5. Badanie anomalii ścieżek

Analizując ścieżki obsługi zgłoszeń wielu projektów *open source* i komercyjnych, zaobserwowałem różne anomalie, które wymagają dokładniejszych badań. Opracowałem schematy analiz, które w zależności od celu badań, charakteryzują poziomy przepływów lub struktury ścieżek. Poziomy przepływów określają przepustowość ścieżek (liczbę obsłużonych zgłoszeń przez daną ścieżkę). Profil ten określa dominujące oraz rzadziej występujące ścieżki. Badania struktury ścieżek dzielę na dwa rodzaje analiz:

1) Analizę długości ścieżek (ścieżki z małą liczbą stanów bez powtórzeń, ścieżki złożone długie z możliwymi powtórzeniami stanów bądź ich pętlami)

2) Analizę ścieżek z określonymi stanami bądź sekwencjami stanów. W sekwencjach stanów możemy wyróżnić pewne ich schematy np.:
*{init, *, term}, {*,seq,*}, {init,*,seq,*,term},{*,term}, {init,*}, {*,seq,*,term}, {init,*,seq, *}, {*, not term}, itp lub {regexp}.*

Gdzie *{}* określa ścieżkę, *init* – stan początkowy, *seq*- stan pośredni, *term* – stan terminalny. Możemy wskazać dowolną kombinację stanów *int*, *seq*, term, co więcej w analizie możemy wykorzystać wyrażenia regularne określające przyjęte warunki przeszukiwania ścieżek. Przykładowo *{'(.*Resolved.*){3,}'}* określa ścieżkę z co najmniej trzema ustawieniami stanu *Resolved*, *{'^[^,\\n]*((,[^,\\n]*){1}$)'}* ścieżka posiadająca tylko dwa zgłoszenia, *{ '^(?=.*Reopened)(?!.*Resolved).+$'}* ścieżka z ustawionym stanem *Reopened,* ale bez stanu *Resolved*. Formuły te określają tak zwane ścieżki kompatybilne, tzn. pozwalają na agregację ścieżek spełniających te same założenia. Tab. 22 przedstawia przykładowe ścieżki kompatybilne w projekcie MongoDB:

Ścieżka kompatybilna	U	N	L Q2	L Q2	AVG T	MAX T	Q2 T	Q3 T
Open...Closed	20	27	5	7,25	49,4	56,3	50,1	53,8
Needs Verification...	1046	12360	9	11	71,3	443,8	52,5	88,8
...In Code Review...	659	8703	10	12	39	429,5	22,3	43,4
Needs Scheduling...In Code Review...	79	791	9	10,5	30	248,6	17,7	30,8
Needs Verification...In Code Review...	544	7858	10	12	39,7	450,1	22,5	44,5

Tab. 22 Przykład prostych ścieżek kompatybilnych projektu MongoDB (liczba zgłoszeń 13435).

Kolumna U określa liczbę unikalnych ścieżek spełniających warunki ścieżki kompatybilnej, N liczba obsłużonych zgłoszeń, L Q2 – mediana długości zgrupowanych ścieżek, L Q3 – trzeci kwartyl długości zgrupowanych ścieżek, AVG T – średni czas obsługi ścieżek, MAX T – maksymalny czas obsługi ścieżek, Q2 T – mediana czasu obsługi ścieżek, Q3 T – trzeci kwartyl obsługi ścieżek.

Bardziej zaawansowane możliwości grupowania ścieżek kompatybilnych daje zastosowanie wyrażeń regularnych, których przykład przedstawia Tab. 23.

Wyrażenie regularne	U	N	P%	Q2 T	Q3 T
^(?!Open).* - Nie zaczynające się od stanu Open	1217	13404	99,77%	7,8	63,9
^[^,\n]*((,[^,\n]*){2}$) – ścieżki 3 stanowe	22	1161	8,64%	5,1	12,9
^[^,\n]*((,[^,\n]*){1}$) – ścieżki 2 stanowe	7	283	2,11%	0,7	3,3
^Open,Closed$ - ścieżki Open,Closed	1	2	0,01%	0,1	0,1
^(?!.*Resolved).+$ - ścieżki bez stanu Resolved	1241	13435	100,00%	7,8	64
^(Needs Scheduling).*(Blocked)+.* - ścieżka rozpoczynające się od stanu Needs scheduling z minimum 1 stanem Blocked	23	27	0,20%	39	275

Tab. 23 Ścieżki kompatybilne zdefiniowane przy użyciu wyrażeń regularnych projektu MongoDB (liczba zgłoszeń 13435)

Kolumna P% określa procent pokrycia ścieżki przez zgłoszenia. Przedstawione wyniki obejmują 13435 zgłoszeń projektu MongoDB (zgłoszenia otwarte i zamknięte typu: *Bug, Improvement, New Feature*).

Ścieżka kompatybilna	U	N	L Q2	L Q3	AVG T	MAX T	Q2 T	Q3 T
N...V/C	287	1910	10	14	42,4	245,1	26,5	45,2
New...	379	2139	9	13	45,4	229,2	30,9	49,3
B-NR...	28	232	5	7	62,4	163,4	52,8	75,2
...NInf	12	18	8	10,25	72,4	83,5	71	74,9
...IPR	19	49	7	10	93,2	183,4	85,6	96
...BLO	11	17	5	6,5	83,6	120,3	78	109
...Reo...	199	370	12	15	77,4	146	68,7	86,3

Tab. 24 Przykład profili prostych ścieżek kompatybilnych projektu P1 (2378 zgłoszeń)

Wyrażenie regularne	U	N	P%	Q2 T	Q3 T
^(?!Open).* - Nie zaczynające się od stanu Open	413	2378	100,00%	13,2	55,1
^[^,\n]*((,[^,\n]*){2}$) – ścieżki 3-stanowe	20	214	9,00%	3,1	45,7
^[^,\n]*((,[^,\n]*){1}$) – ścieżki 2-stanowe	8	148	6,22%	5,8	20,5
^Open,Closed$ - ścieżki Open,Closed	24	153	6,43%	5,8	23
^(?!.*Resolved).+$ - ścieżki bez stanu Resolved	384	2048	86,12%	8,8	63,8
^(?=.*Reopened)(?!.*Resolved).+$ - ścieżki ze stanem Reopened bez Resolved	406	750	31,54%	34,7	132,8

Tab. 25 Ścieżki kompatybilne zdefiniowane przy użyciu wyrażeń regularnych projektu P1 (2378 zgłoszeń)

Tab. 24 i Tab. 25 zawierają przykłady ścieżek kompatybilnych wygenerowanych dla projektu P1 uwzględniającego 2378 zgłoszeń (zgłoszenia otwarte i zamknięte typów *Bug, New Feature, User Story, Task*).

W koncepcji IHG możemy wyróżnić ścieżki pełne oraz kompatybilne. Ścieżki pełne uwzględniają wszystkie stany, poczynając od stanu początkowego, do ostatniego stanu ustawionego w repozytorium dla danego zgłoszenia. Ścieżki kompatybilne definiujemy przez *prefix* (stan początkowy), *sufix* (stan końcowy) lub sekwencję stanów (włączając stan wewnętrzny). Ścieżki kompatybilne odnoszą się do szczegółowej analizy procesu obsługi zgłoszeń, a w szczególności wykrywania anomalii obsługi zgłoszeń jak np. ścieżki 2-stanowe *Open,Closed* lub kilkukrotne wystąpienie stanu *Closed.*

Wyszukując anomalie procesu obsługi zgłoszeń, możemy wykorzystywać proste formy, określające np. długości ścieżek bądź konkretnej sekwencji statusów np. *Open, Closed*. Możemy też wykorzystywać bardziej złożone wyrażenia regularne określające sekwencję stanów końcowych bądź duplikację stanów. Po znalezieniu „niepokojących" ścieżek możemy przystąpić do szczegółowej analizy poszczególnych zgłoszeń, identyfikując źródło wystąpienia anomalii. Szczegółowe analizy powinny zostać skorelowane z komentarzami bądź reporterami lub osobami uczestniczącymi w obsłudze zgłoszenia dla szerszej perspektywy.

Niezależnie od wielkości projektu, każdy dobry proces obsługi zgłoszeń powinien zaczynać się od jednego zdefiniowanego stanu początkowego. Analizując szereg repozytoriów zaobserwowałem zmienność stanu początkowego lub występowanie wielu stanów początkowych procesu obsługi zgłoszenia. Przykłady stanów początkowych zgłoszeń błędów o różnych priorytetach przedstawiłem w Tab. 26 (MongoDB) i Tab. 27 (P1). Parametry stanów początkowych zaprezentowane w tabelach definiowane są poprzez krotkę: {*Liczba unikalnych ścieżek rozpoczynających się od stanu S; Liczba zgłoszeń, które zaczęły ścieżkę od stanu S*}. Jeżeli stan nie występuje jako początkowy parametry pozostały puste dla odpowiedniego wiersza. Szczególnie duża różnorodność stanów początkowych widoczna jest w projekcie MongoDB, gdzie dla zgłoszeń błędów występuje aż 7 stanów początkowych. Świadczy to o nieporządku w procesie obsługi zgłoszeń oraz jego zmienność na przestrzeni badanego okresu. W porównaniu z projektem komercyjnym jest on nadal w fazie stabilizacji. Poprzez fazę stabilizacji rozumiem okres, w którym zespół projektowy w ramach synergii optymalizuje proces obsługi zgłoszeń.

W projekcie komercyjnym występują dwa stany początkowe zgłoszeń błędów *N (New), B-NR (Backlog – Not Ready)*, z dominacją stanu N przez który przechodzi 99,8% zgłoszeń.

Stan *B-NR* pojawia się w ścieżkach obsługi zgłoszeń błędów, z uwagi na zmianę typu zgłoszenia podczas jego procesowania w repozytorium. Trzy zmienione zgłoszenia pierwotnie został dodane do repozytorium jak zgłoszenia typu *User Story*, po czym nastąpiła ich konwersja do typu błąd (*Bug*). Kwestia ta może zostać uznana za anomalię procesu. Analogiczna sytuacja występuje dla innych typów zgłoszeń, które tworzone były poprzez transformację zgłoszeń typu *User Story* na *New Feature* i odwrotnie. Z uwagi na brak dostępu do szczegółowych danych repozytorium JIRA dla projektu MongoDB nie jestem w stanie, uzasadnić tak wysokiej różnorodności stanów początkowych zgłoszeń błędów.

Priorytet	NV	NES	NTR	O	REQ	BAC
Blocker	12;51	3;3				
Critical	39;115	4;4	2;2			
Major	527;6840	63;581	33;52	11;13	1;1	1;1
Minor, Trivial	65;215	4;8	7;7	1;1		

Tab. 26 Stany początkowe zgłoszeń zamkniętych błędów projektu MongoDB – Liczba unikalnych ścieżek rozpoczynających się od stanu S (NV, NES, NTR, O, REQ, BAC); Liczba zgłoszeń, które zaczęły ścieżkę od stanu S

Priorytet	N	B-NR
Blocker	16;32	
Critical	72;270	1;1
Major	154;609	
Minor	101;487	2;2
Trivial	14;33	

Tab. 27 Stany początkowe zgłoszeń błędów projektu P1 – Liczba unikalnych ścieżek rozpoczynających się od stanu S (N, B-NR); Liczba zgłoszeń, które zaczęły ścieżkę od stanu S

W procesie obsługi zgłoszeń można zauważyć pętle stanów. Pętle te są często kłopotliwe do identyfikacji z uwagi na ich różnorodność, wspieram taką analizę opracowanym algorytmem (Alg. 12). Definiuję w nim rząd anomalii jako długość cyklu zapętlenia oraz powiązuje je z pewnymi danymi rozważanej ścieżki (czas obsługi, liczba zgłoszeń które ulegają zapętleniu). Rząd pętli wielkości 1, świadczy o zapętleniu pojedynczego stanu. Może być to powiązane z błędem narzędzia JIRA lub całkowitym braku formalizacji procesu. Sytuacja ta występowała jednak bardzo rzadko w badanych repozytoriach.

Anomalie typu pętla w ścieżce w badanych projektach mają swój procentowy udział w całkowitej liczbie zgłoszeń między 0,02% - 5,34%. Szczególnie istotne wydają się anomalie pierwszego rzędu, w których zapętlony jest jeden stan. Pętle jednostanowe odkryłem w dwóch

projektach MongoDB oraz Apache Lucene. W przypadku projektu MongoDB procent anomalii (pętli ścieżek) wynosił 4,16% (badanych zgłoszenia błędów), 3,22% (badanych zgłoszenia nowych funkcjonalności). Wśród tych zgłoszeń ponad połowa wykrytych anomalii dotyczyła pętli w tym samym stanie (I rząd anomalii). W projekcie Apache Lucene 2,46% (błędów), 2,70% (nowych funkcjonalności) zgłoszeń zawierało w sobie pętle statusów z czego odpowiednio 81,63% i 78,74% zgłoszeń z anomaliami zawierało pętlę w tym samym stanie. Wysoki procentowy udział pętli pierwszego rzędu we wszystkich zgłoszeniach z anomaliami budzi pewne wątpliwości co do stabilności procesu obsługi zgłoszeń. Sprawa ta powinna zostać zweryfikowana ręcznie przez eksperta projektu, w celu odnalezienia źródła tego problemu.

Tab. 28 i Tab. 29 przedstawiają przykłady ścieżek z anomaliami oraz największym pokryciem zgłoszeń dla projektów odpowiednio MongoDB oraz Projektu P1. W projekcie MongoDB dla 8405 zgłoszeń błędów anomalie wystąpiły w 350 zgłoszeniach, dla projektu komercyjnego anomalie wystąpiły w 83 z1555 zgłoszeń błędów. Tabele uwzględniają średni, maksymalny, medianę oraz 3 kwartyl czasu obsługi ścieżki podanej w kolumnie *ścieżka*, pętle oznaczone są wytłuszczonym tekstem. Długość całej ścieżki podana jest w kolumnie *L*, Kolumna *A** reprezentuje liczbę wykrytych pętli. Przykładowo ścieżka *NED,INV,WFU,INV,WFU,INV,WFU,INV,CLO* zawiera dwie pętle: *INV,WFU,INV,WFU,INV,WFU,INV* oraz *WFU,INV,WFU,INV,WFU* które zostały wyznaczone zgodnie z opracowanym algorytmem (Alg. 12). Kolumna *N* uwzględnia liczbę zgłoszeń które przeszły przez ścieżkę.

W projekcie komercyjnym P1 najczęściej występują pętle dotyczące 3 stanów, co istotne pętle inicjowane są w większości przypadków przez programistów. W anomaliach oraz ustawianych statusach zgłoszeń widoczny jest jasny podział odpowiedzialności. Stąd zapętlenia zaczynające się od stanów jak *IN PROGRESS (IN), To be Tested (TbT)* inicjuje developer zaś od statusu *New* (*N*) bądź *REOPENED* (*REO* - nie ma w tabeli) inicjuje tester. Pętli inicjowanych przez testerów jest o wiele mniej w porównaniu do pętli rozpoczętych przez programistów. W projekcie pojawia się bardzo mało unikalnych ścieżek z tymi samymi stanami w anomaliach. Większość z nich są to unikalne ścieżki, przez które przechodzi tylko jedno zgłoszenie (58 z 66 unikalnych ścieżek z anomaliami). Występujące anomalie są standardowe jednak wskazują na niedoskonałości w dostarczaniu jakościowego rozwiązania. Problem widoczny jest w szczególności kiedy w zapętleniu pojawia się status *Reopened*. Jasne jest wtedy, że dostarczona poprawka nie zadziałała. Anomalie nie wykazują powiązania z liczbą osób zaangażowanych w proces bądź liczbą komentujących osób. Istotny jest również fakt braku anomalii

w zgłoszeniach błędów o najwyższym priorytecie, co jest dobrym wyznacznikiem jakości obsługi takich raportów. Badania Anomalie mogą dotyczyć także innych zagadnień np. powtórzeń stanu *Resolved_X* w ścieżce obsługi zgłoszenia, gdzie X oznacza różne sposoby rozwiązania zgłoszenia (ang. *resolution*), przykładowo możliwe są zgłoszenia w których ścieżki zawierają dwa takie same stany np. *Resolved_Fixed* lub dwa różne stany np. *Resolved_Fixed* oraz *Resolved_Duplicate*. Takie anomalie możemy śledzić z wykorzystaniem opracowanych ścieżek kompatybilnych uwzględniających wzorce regularne.

N	Ścieżka	L	AVG	MAX	Q2	Q3	A*
44	NED,NES,OPE,IPR,ICR,**CLO,CLO**	7	43,1	366,2	11	37,3	1
37	NED,**INV,WFU,INV,WFU,INV,**CLO,CLO	7	43	240,9	32,3	49,1	1
12	NED,**INV,WFU,INV,WFU,INV,WFU,INV**,CLO	9	113,5	545,1	59,4	133,3	2
11	NES,OPE,**NES,NES**,OPE,IPR,ICR,CLO	8	13,7	35,2	11	15,5	1
9	NED,**INV,WFU,INV,WFU,INV,WFU,INV,WFU**,CLO	10	112	281,4	91,2	117,6	2
8	NED,NES,OPE,IPR,ICR,NEM,**CLO,CLO**	8	53,1	133	49,9	70	1
8	NED,**INV,WFU,INV,WFU,INV,WFU**,CLO	8	133,6	246,8	112	196,6	2
5	NED,NES,**CLO,CLO**	4	13,1	48,7	5,1	8,3	1
5	NED,NES,**OPE,IPR,OPE,IPR,OPE**,IPR,ICR,CLO	10	76,4	137,3	97,7	120	2

Tab. 28 Przykłady ścieżek obsługi zgłoszeń błędów projektu MongoDB z anomaliami i największym pokryciem zgłoszeń

N	Ścieżka	L	AVG	MAX	Q2	Q3	A*
9	N,**IN,REV,TbT,REO,IN,REV,TbT, REO,IN**,**REV,TbT**,V/C	13	86,5	213,1	32,8	133,2	3
4	N,**IN,NIF,IN,NIF,IN,**REV,TbT,V/C	9	41,9	67,8	42,6	58,4	1
2	N,**IN,REV,TbT,REO,IN,REV,TbT,REO,IN,REV, TbT,REO,IN,REV,TbT,REO,IN,REV,TbT**,V/C	21	154,1	252,2	154,1	252,2	14
2	**N,IN,N,IN,N,IN**,REV,TbT,V/C	9	42,4	74	42,4	74	2
2	N,**IN,REV,TbT,REO,IN,REV,TbT,REO,IN,**NIF,IN, REV,TbT,V/C	15	25,7	39	25,7	39	1
2	**N,IN,N,IN,N,IN,N,IN**,REV,TbT,V/C	11	64,2	69,2	64,2	69,2	4
2	N,**TbT,REO,TbT,REO,TbT**,V/C	7	53,5	105,1	53,5	105,1	1
2	N,**IN,REV,IN,REV,IN,**N,IN,REV,TbT,V/C	11	156,5	175	156,5	175	1
1	N,IN,REV,TbT,V/C,REO,IN,REV,**TbT,V/C,REO,TbT, V/C,REO,TbT,V/C,REO,TbT**,REO,V/C	20	209	209	209	209	4

Tab. 29 Przykłady ścieżek obsługi zgłoszeń błędów projektu P1 z anomaliami i największym pokryciem zgłoszeń

5.2.6. Analiza czasowa obsługi zgłoszeń w ścieżkach

Przedstawione wcześniej charakterystyki czasowe (sekcja 5.1.) pokazują perspektywę ogólną czasów obsługi poszczególnych zgłoszeń. Dane te mogą służyć do wyciągania generalnych wniosków na temat procesu, jednak dla lepszego zrozumienia zachodzących w nim zdarzeń potrzebne są analizy szczegółowe (drobnoziarniste) z pewnym poziomem agregacji skorelowanych ze sobą zgłoszeń. Opracowany model IHG pozwala na korelację zgłoszeń względem ścieżki obsługi, którą przechodzą zgłoszenia. Czasy przetwarzania ścieżek przedstawiam w perspektywach: średniej, maksymalnej, kwartyla Q2 i Q3. Przykład takiej agregacji dla projektu MongoDB przedstawia Tab. 30.

N	Ścieżka	L	AVG	MAX	Q2	Q3
22	NES,OPE,NES,ICR,NEM,CLO	6	12,8	83,2	1,4	5,3
112	NES,OPE, NES,IPR,ICR,NEM,CLO	7	8,2	144,4	1,8	5,7
9	NED,NES,OPE,ICR,CLO,OPE,CLO	7	7,9	42,9	2,2	6,5
144	NES,OPE,NES,ICR,CLO	5	9,7	216,4	2,4	7,1
674	NED,NES,OPE,ICR,CLO	5	12,8	372,8	4	10,9
102	NED,NES,OPE,ICR,NEM,CLO	6	14,7	191,9	4,3	13,8
367	NES,OPE,NES,IPR,ICR,CLO	6	20,4	342,7	5,2	17,9
235	NED,NES,INV,IPR,ICR,CLO	6	14,6	224,7	5,4	13,9
54	NES,OPE,NES,CLO	4	64,1	723,2	5,6	33,2
29	NED,NES,INV,IPR,CLO	5	16,9	82,1	5,9	29,8

Tab. 30 Przykłady ścieżek obsługi zgłoszeń projektu MongoDB, posortowane rosnąco względem kwartyla Q2 czasu

Profil unikalnych ścieżek pozwala na korelację długości ścieżki z jej czasem obsługi oraz sprawdzenie możliwych trendów, generowane profile modelu IHG uwzględniają również perspektywę liczby dodawanych komentarzy. W projekcie komercyjnym zauważalna jest tendencja do wzrostu trzeciego kwartyla wraz ze wzrostem liczby komentarzy. Maksymalna wartość kwartyla Q3 czasu obsługi ścieżki dla zgłoszeń błędów (bez ścieżek z pokryciem poniżej 2 zgłoszeń) powiązana jest z najwyższą średnią liczbą komentarzy. Trzy ścieżki z najwyższym czasem Q3 możemy scharakteryzować przez wektory {289,4; 8; 10; 3}, {238,7; 5; 5;4}, {226,2;3;9;5}, gdzie kolejne liczby reprezentują odpowiednio trzeci kwartyl czasu, średnią liczbę komentarzy, długość ścieżki, liczbę zgłoszeń przechodzących przez ścieżkę. Interesujące są również profile ścieżek z ograniczenia czasu np. kwartla Q3, określające procent zgłoszeń, które zostały obsłużone poniżej zadanej wartości kwartyla Q3. Przykładowo,

w weryfikowanych zgłoszeniach błędów projektu komercyjnego 74% z 1555 zgłoszeń zostało obsłużonych poniżej Q3= 50. Na tej podstawie wyznaczyłem pewną miarę badania zakresów obsługi profili czasowych do weryfikacji w ich różnych zakresach. Krotkę perspektywy definiuję przez 3 wartości:

$$TP_i = \{\Delta_i, n_i, i_i\}, \Delta_i = a_i - b_i \quad (5.12)$$

Gdzie Δ_i określa zakres dni dla weryfikowanego profilu czasu TP (np. Q2,Q3, AVG, MAX), n_i oraz i_i określa odpowiednio liczbę ścieżek i zgłoszeń powiązanych z zakresem i. Index i jest rosnącą wartością iterującą po kolejnych zakresach czasu. Profile te mogą być najbardziej interesujące przy wyznaczaniu dominujących ścieżek w specyficznych zakresach. Dla przykładu wyznaczyłem kolejne profile TP Q3 dla zgłoszeń błędów projektu MongoDB, w którym przetworzono łącznie 8405 zgłoszeń.

$Q2_1$={1-5, 57,19,58%}, $Q2_2$={5-14,114, 57,63%}, $Q2_3$={15-30, 139, 5,37%}

$Q3_1$ ={1-6, 70, 9,92%}, $Q3_2$={7-14, 59, 14,77%}, $Q3_3$ ={14-30, 132, 46,19%}

Maksymalny czas obsługi zgłoszeń błędów w projekcie MongoDB przyjmował wartości od 1h do 3 lat. Znacząca część zgłoszeń (70,88%) była obsłużona z czasem Q3 mniejszym niż 30 dni. W procesie obsługi zgłoszeń występują także anomalie maksymalnego czasu obsługi ścieżki zgłoszenia. W repozytorium dostępne są 3 zgłoszenia błędów (priorytet Major) przechodzące przez ścieżkę *NV, NSch, O, Bac, Nsch, Bac, C* (7 stanów) z maksymalnym czasem obsługi 3 lata.

Dla zgłoszeń błędów projektu Arrow 66,99% zgłoszeń zostało przeprocesowanych dla Q3 < 1 m-c, natomiast dla Q2 < 1m-c było to już 83,56% zgłoszeń. Gorsze czasy obsługi zaobserwowałem w projekcie Apache Lucene, w którym dla Q3 < 1m-c rozwiązano jedynie 15,16% zgłoszeń błędów zaś dl Q2<1m-c wartość wzrasta do 22,95%. Zauważalne jest jednak pogorszenie procesowania zgłoszeń w projekcie Apach Lucene w porównaniu z MongoDB bądź Arrow.

Dla projektów prowadzonych w metodyce SCRUM istotniejszym wskaźnikiem obsługi czasu jest procent przesuniętych zgłoszeń między sprintami. Dla badanego projektu P1 dla 10 następujących po sobie sprintach następowało przesunięcie 6.8%, 6.2%, 15.6%,11%, 10%, 6.8%, 15%, 6.5%, 10%, 3.1% zadań do następnego sprintu tzn. zgłoszenie nie zostało dostarczone w sprincie n i zostało przeniesione do sprintu n+1. Ścieżki modelu IHG prezentują zagregowany czas obsługi zgłoszeń. W projektach komercyjnych opartych o *SCRUM* zgłoszenia nowych funkcjonalności trafiają najczęściej do rejestru zadań przeznaczonych do priorytetyzacji

(ang. *Backlog*) przez Właściciela Produktu. Gdzie oczekują na dodanie do sprintu w zależności od potrzeb biznesowych. W przedstawionym podejściu czas obsługi zgłoszenia liczony jest od chwili dodania raportu do repozytorium, przez co czas w odniesieniu do sprintów *scrumowych* może być mylący. Z drugiej strony pokazuje to także problem oczekujących zgłoszeń, którego główną przyczyną może być zbyt mały zespół projektowy na zgłoszoną liczbę zadań.

5.2.7. Dyskusja

Analiza wielu repozytoriów projektowych ujawniła potrzebę bardziej wnikliwych badań, ukierunkowanych na ocenę jakości procesu obsługi zgłoszeń. Ręczna eksploracja takich repozytoriów może być żmudna i pracochłonna. Dostępne narzędzia umożliwiają ocenę poszczególnych zgłoszeń, jak np. czas stworzenia zgłoszenia, czas jego obsługi (poprzez porównanie czasu dodania zgłoszenia do repozytorium z czasem jego rozwiązania), sposobu rozwiązania itp. Do lepszego zrozumienia procesu niezbędne są pewnego rodzaju agregacje, które wspiera przedstawiony model IHG. Analizy te wspierane są przez zaimplementowane narzędzie *IssueAnalyzerTool* oraz szereg skryptów ułatwiających prezentację omówionych perspektyw. Generowane charakterystyki przedstawiają miedzy innymi czasy obsługi (MIN, AVG, MAX, kwartyl Q1, Q2, Q3) w perspektywie ścieżek zgłoszeń, dystrybucje zgłoszeń między ścieżkami, stanami. Perspektywy te pokazują dominujące ścieżki obsługi zgłoszeń, ale również najrzadziej występujące stany.

Model IHG umożliwia również wprowadzenia ograniczeń perspektyw do określonego m.in.: typu zgłoszenia, istotności, grupy reporterów bądź okresu w jakim zgłoszenia zostały dodane w repozytorium. W przeanalizowanych repozytoriach zaobserwowałem przypadki, kiedy obsługa błędów zgłoszeń o najwyższej priorytecie była o wiele lepsza niż reszty zgłoszeń. Obserwacja widoczna jest dla większości projektów. Potwierdza to zasadność użycia pola Priorytet. Z badań w rozdziale 3 (Tab. 3) wynika że poziomy priorytetów często nie są dobrze przypisywane zgłoszeniom (np. dominuje priorytet Major). Istotne jest zatem przeprowadzanie analiz od szczegółu do ogółu w celu identyfikacji najbardziej newralgicznych punktów procesu. Przedstawione analizy pokazują także sposoby oceny raportowania zgłoszeń oraz wyszukiwania anomalii procesu. Użyteczne w tej aktywności są opracowane ścieżki kompatybilne ze zdefiniowanymi stanami początkowymi, pośrednimi bądź terminalnymi. Zaimplementowane narzędzie *IssueAnalyzerTool* pozwala także na wykorzystanie wyrażeń regularnych w celu identyfikacji bardziej skomplikowanych ścieżek kompatybilnych. Dzięki wyrażeniom regularnym możemy określić pewne niedoskonałości procesu obsługi zgłoszeń, przykładowo ile zgłoszeń miał ustawiony co najmniej 2 razy stan *IN PRGOGRESS*,

jednak bez wystąpienia stanu *RESOLVED*. Anomalie procesu mogą dotyczyć także zapętleń stanów, które wyznacza algorytm (Alg. 12). W analizowanych projektach anomalie procesu w postaci pętli stanów dotyczyło między 0,02%, a 5,34% wszystkich zgłoszeń. Najbardziej krytyczne anomalie I rzędu (pętle w tym samym stanie) wykryłem w projekcie MongoDB dla 2,13% zgłoszeń błędów (179). Ścieżki procesujące zgłoszenia z takimi anomaliami stanowiły 10,58% wszystkich zweryfikowanych ścieżek. Podobne anomalie wystąpiły również w zgłoszeniach projektu Apache Lucene z rozkładem 2,01% (procent zgłoszeń z zapętleniem 1-stanowym), 2,46% (procent zgłoszeń ze wszystkimi pętlami) dla zgłoszeń błędów (3983) oraz dla zgłoszeń nowych funkcjonalności (4702) odpowiednio 2,13%, 2,70%. Występowanie zapętleń jednego statusu jest niepokojące i powinno zostać zweryfikowane ręcznie przez eksperta z zespołu projektowego np. managera. Anomalie 1-stanowe nie występują w projekcie komercyjnym P1. Występowanie anomalii procesu obsługi zgłoszeń w postaci pętli stanów, jest często tożsame z powtarzaniem analizy co prowadzi do wydłużonego czasu obsługi zgłoszenia. Dlatego ważne jest stałe monitorowanie procesu obsługi zgłoszeń oraz identyfikacje takich niedoskonałości.

Opracowany model IHG pozwala także na przeprowadzanie porównań procesów obsługi zgłoszeń w różnych projektach lub kolejnych wersjach tego samego systemu. Może być to wartościowe przy ocenie poszczególnych projektów oraz wyciągnięcia wniosków wspierających optymalizację procesów obsługi zgłoszeń w przypadku zestawienia projektu o dobrej kondycji procesu względem projektu z niedoskonałościami.

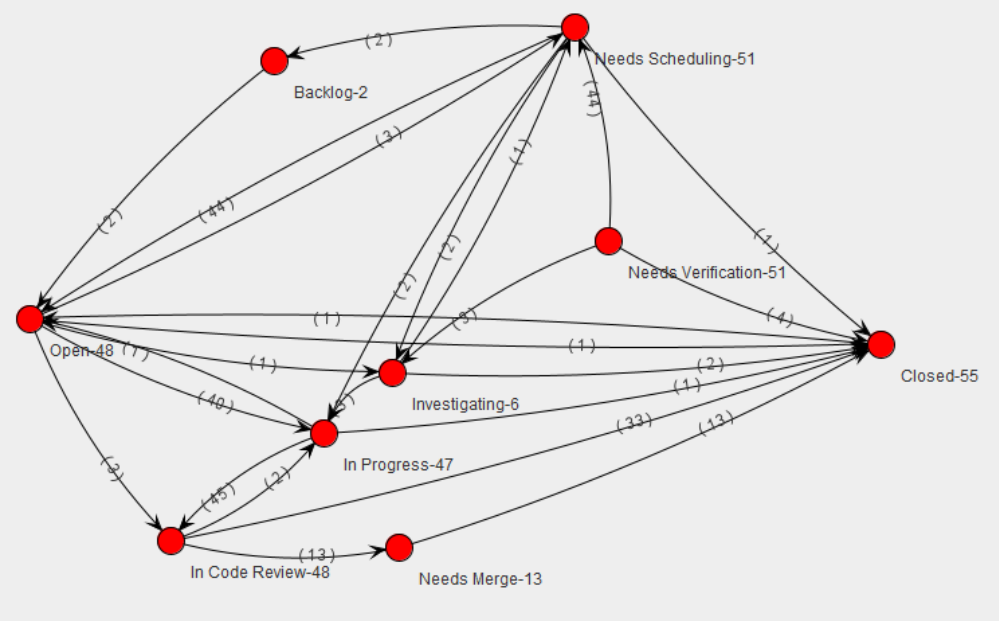
Porównując dwa grafy G1 i G2 należy dokonać pewnych normalizacji poprzez wprowadzenie spójnych nazw stanów oraz funkcji etykietowanych np. względne przepływy zgłoszeń (przepustowość). Takie procesy porównawcze mogą dostarczyć różnych kategorii grafów: 1) grafy dla projektów tej samej klasy realizowanych w podobnym środowisku programistycznym (np. kolejne wersje projektu), 2) grafy tego samego projektu związane z różnymi ograniczeniami R, 3) grafy projektów realizowanych w różnych środowiskach programistycznych. Możliwe jest również porównanie zredukowanych grafów lub podgrafów zgodnie z celami analizy.

Porównania mogą być ukierunkowane na cechy strukturalne, przepływowe i czasowe. Wyniki porównań są przydatne do znalezienia różnic, a następnie oceny ich wpływu na procesy obsługi zgłoszeń lub anomalie postępu projektu w porównaniu z innymi projektami. Przydatne jest wprowadzenie pewnych miar odmienności. Na przykład porównując wektory przepływu stanów *sf* dwóch grafów, możemy obliczyć odległość euklidesową (ED) lub

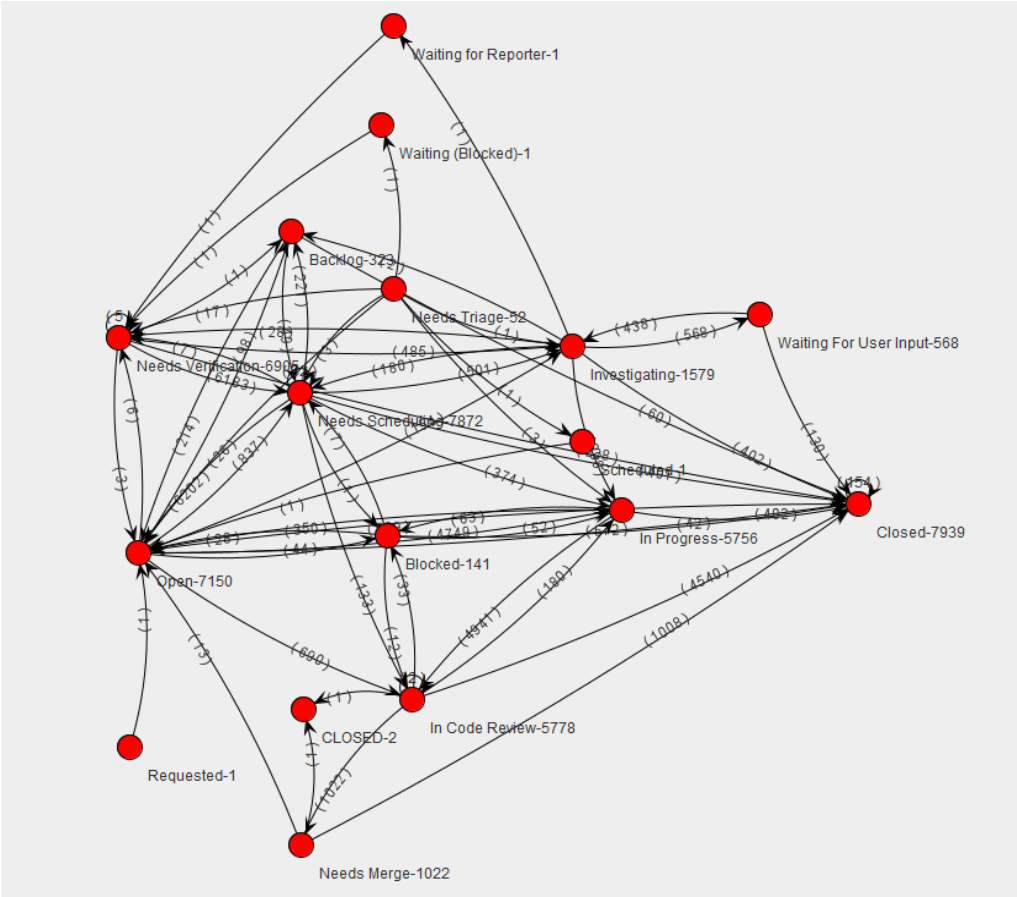
różnicę całkowitą (TD), obliczoną jako suma różnic bezwzględnych po wszystkich elementach wektora (np. stanach) [14]. Tutaj powinniśmy użyć przepustowości względnych ze względu na różną liczbę zgłoszeń w projektach. W analizie szukamy podobieństw i różnic w obsłudze zgłoszeń. Zidentyfikowane odchylenia od modelu referencyjnego można badać w sposób systematyczny w celu znalezienia przyczyn ich występowania i możliwych usprawnień w procesach obsługi zgłoszeń. Taka analiza może być dostosowana do jej celów, poprzez zastosowanie różnych ograniczeń R modelu IHG np. typ zadań, ich priorytet lub zaangażowanych aktorów (np. użytkowników, deweloperów) itp. Ostateczna analiza może zostać przeprowadzona przez ekspertów zarządzania projektami i przedyskutowana podczas spotkań retrospektywy SCRUM z uczestnikami projektu.

Dla zobrazowania metody porównań modeli IHG wygenerowałem dwa grafy:
1) G_1 MongoDB z ograniczeniem R - zamknięte zgłoszenia błędów o priorytecie *Blocker* G_1<9;25> , d(G) =0,35, indeg(V) = (0-7), outdeg(V) = (1-5)
2) G_2 MongoDB z ograniczeniem R - zamknięte zgłoszenia błędów o priorytecie *Major* G_2<17;66> , d(G) = 0,24, inden(V) = 0-11, outdeg(V) = 0-9.

W G_1 i G_2 uwzględnionych zostało odpowiednio 54 i 7488 zgłoszeń. Pełen graf G_1 przedstawia Rys. 6 a jego zredukowana forma dla istotności wierzchołków i krawędzie większej niż 20% Rys. 8. Graf G_2 w tej samej notacji przedstawiają Rys. 7 i Rys. 9. Co ciekawe w przypadku obydwu grafów widoczna jest znaczna redukcja stanów i krawędzi dla zastosowanego ograniczenia 20% odpowiednio z 9 do 7 dla G_1 i z 17 do 6 dla G_2.

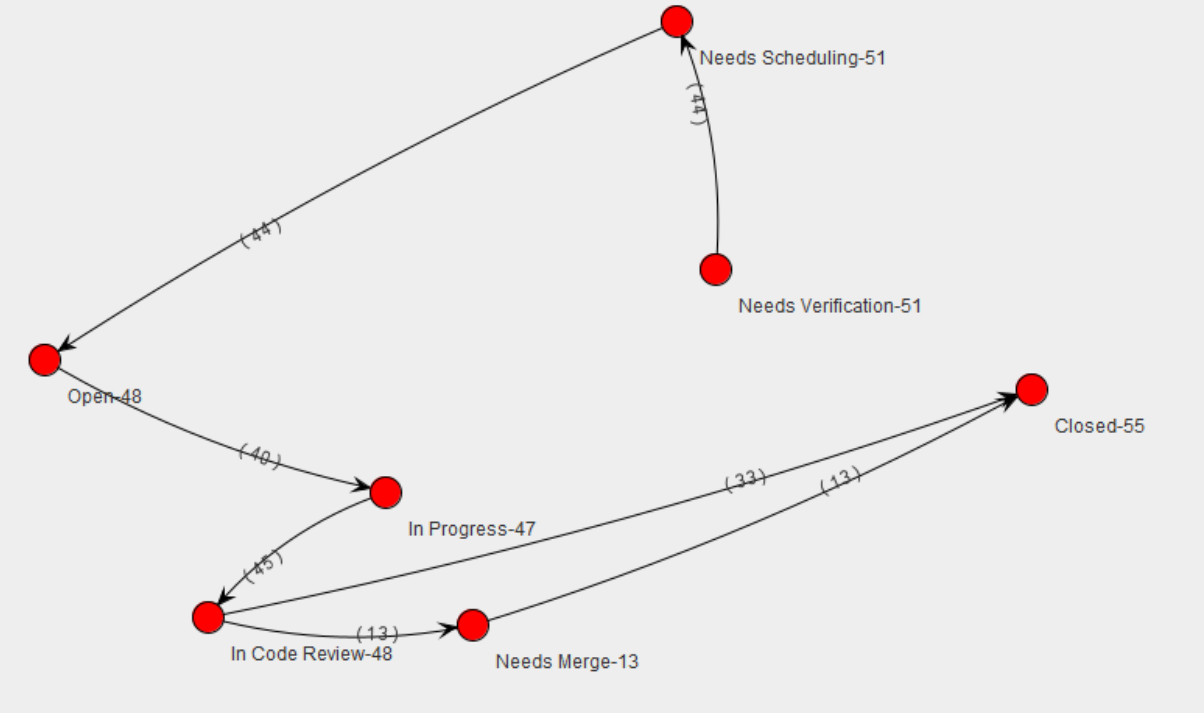


Rys. 6 Graf IHG projektu MongoDB z ograniczeniem R – zamknięte zgłoszenia błędów o priorytecie Blocker (54 zgłoszenia)

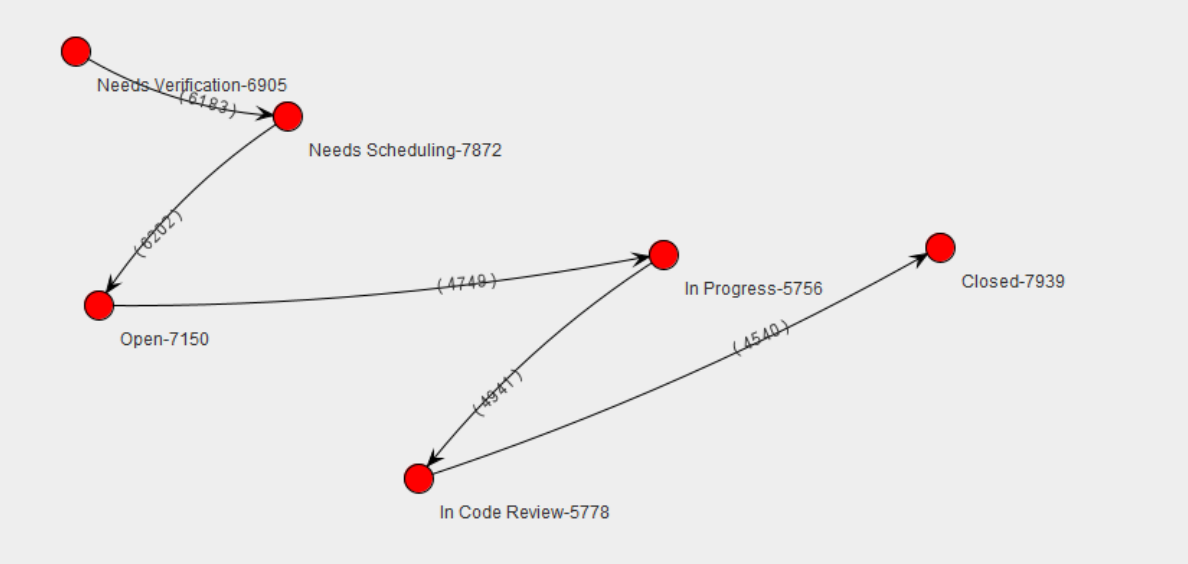


Rys. 7 Graf IHG projektu MongoDB z ograniczeniem R - zamknięte zgłoszenia błędów o priorytecie Major (7488 zgłoszeń)

Rys. 7 przedstawia wyższy poziom skomplikowania grafu G_2. Dla obydwu modeli występuje ta sama ścieżka dominująca *Needs Verification,Needs Scheduling, Open, In Progress, In Code Review, Closed* bez anomalii, z wartościami profili przedstawionymi w formie wektora {46,30%;6;8;39,3;4,1;7,1;2} oraz {34,71%;6;18,5;511,5;5,5;18,4;2} odpowiednio dla priorytetu *Blocker* i *Major*. Wartości opisujące wektor reprezentują procent pokrycia ścieżki przez wszystkie zgłoszenia grafu, czas obsługi zgłoszenia przez ścieżkę (w dniach): AVG, MAX, Kwartyl Q2,Q3. Ostatni parametr określa średnią liczbę komentarzy w zgłoszeniach ścieżki. Zauważalny jest wzrost czasów obsługi ścieżki dominującej dla zgłoszeń o niższym priorytecie Kwartly Q2 czasu obsługi wzrósł z 7 dni dla *Blocker* do 18 w zgłoszeniach z priorytetem *Major*.



Rys. 8 Zredukowany graf IHG z ograniczeniem R – zamknięte zgłoszenia błędów o priorytecie Blocker (poziom filtracji istotności 20%)



Rys. 9 Zredukowany graf IHG z ograniczeniem R - zamknięte zgłoszenia błędów o priorytecie Major (poziom filtracji istotności 20%)

Zredukowany graf z Rys. 9 zostały ograniczone do 1 ścieżki, która jest ścieżką dominującą. Wystąpienie stanu *Open* w połowie procesu, świadczy o specyficznej konfiguracji przepływu zgłoszeń błędów w projekcie MongoDB. W analizie porównawczej czasów obsługi stanów widoczne są niższe wartości profili dla zgłoszeń priorytetu *Blocker* (Graf G_1),

profil sr* osiąga w nich również niższe wartości niż w przypadku zgłoszeń priorytetu *Major* (Graf G_2). Przykładowo najwyższa wartość sr* = 0,0784 w G_1 odnosi się do stanu *Need Scheduling*, dla G_2 najwyższa wartość sr* = 0,4137 odpowiada stanowi *Waiting For User Input*, co świadczy o wysokiej redundancji stanu w grafie G_2. W ścieżkach grafu G_2 widoczne są także pętle które nie występują w grafie G_1. Pętle w grafie G_2 stanowią 4,05% wszystkich zgłoszeń z fluktuacją długości pętli 1-9. Świadczy to o większej złożoności oraz niedoskonałościach procesu obsługi zgłoszeń priorytetu *Major* projektu MongoDB (graf G_2).

Kolejnym typem porównań wspieranych przez koncepcję IHG jest zestawienie modeli różnych typów zgłoszeń. Np. zgłoszeń *New Feature* oraz *Bug*. Przykładem takiego porównania może być projekt MongoDB dla którego zgłoszono 773 zgłoszenia typy *New Feature* oraz 8405 zgłoszeń typu *Bug*. Zgłoszenia błędów rozwiązywane są szybciej, z medianą dla ścieżki dominującej równą 5,8 dnia dla porównania mediana czasu obsługi ścieżki dominującej dla zgłoszeń typu *New Feature* wynosi 37 dni. W zgłoszeniach błędów występuje jednak wyższa fluktuacja zarówno czasów obsługi jak i długości ścieżek. Maksymalny czas obsługi ścieżki zgłoszenia błędu wyniósł 1104 dni, w przypadku zgłoszeń typu *New Feature* wartość ta była równa 976 dni. Fluktuacja długości ścieżek wynosiła odpowiednio 1-28 oraz 2-18. Długość ścieżek przekłada się również na liczbę pętli w nich występujących. W przypadku zgłoszeń *New Feature* występują jedynie 23 zgłoszenia z anomaliami obsługi stanów, które przechodzą przez osobne ścieżki (23) z maksymalną liczbą komentarzy równą 24. W przypadku zgłoszeń błędów liczba raportów z anomaliami wynosi 350, przechodzących przez 174 unikalne ścieki z fluktuacją długości 3-22, w których maksymalna liczba dodanych komentarzy wynosiła 54. Widoczna jest korelacja długości ścieżek ze wzrostem czasu ich obsługi (zgłoszenia błędów) oraz złożonością struktury (liczba pętli i komentarz).

Możliwe do przeprowadzenia są również porównania innych grup grafów IHG agregujących ścieżki zgłoszeń np.: raportowanych przez różne grupy aktorów, zgłoszeń z określonymi wartościami atrybutów (komponent, wersja rozwiązania itp.). Interesujące grupy zgłoszeń można określić na podstawie analiz przedstawionych w rozdziale 3.

Istotnym aspektem analizy procesu obsługi zgłoszeń jest perspektywa zaangażowania aktorów na poszczególnych etapach procesu, np. tworzenie zgłoszeń lub obsługa (dostarczenie poprawki, testowanie rozwiązania). Pewien obraz takich analiz przedstawiłem w sekcji 3.3.2, jednak skupiały się one na pierwszej fazie procesu tzn. tworzeniu zgłoszeń. Poniższa dyskusja pokrywa dalsze procesowanie zgłoszeń na podstawie opracowanych grafów IHG. W projekcie komercyjnym (P1) widoczny jest jasny podział obsługi zgłoszeń błędów tzn. testerzy

odpowiadają za tworzenie zgłoszeń błędów oraz przestawiają zgłoszenia w określonych statusach. Analizując zgłoszenia w badanym zakresie, testerzy zmieniali status jedynie z *TbT* na *REO* lub z *TbT* do *V/C*. Pozostałe statusy w większość przypadków są zmieniane przez programistów. Zdarzały się jednak jednostkowe przypadki zmian dodatkowego statusu z *REV* na *TbT*. Związane może to być z komunikacją poza *JIRA* gdzie testerzy są w stałym kontakcie z programistami i otrzymują potwierdzenie o pomyślnej weryfikacji kodu i możliwości przeprowadzenia testów poprawki. W niektórych zgłoszeniach pojawia się także status *NInf,* który związany jest z potrzebą uzyskania dodatkowych informacji. Taki status ustawiany jest przez programistów jednak dalszy przepływ zgłoszenia zależy od informacji dodanej przez Analityka. W przypadku potwierdzenia potrzeby poprawki, zgłoszenie trafia do programisty ze statusem *IPR*. Jeżeli zgłoszenie nie wymaga poprawki jest ono zamykane z odpowiednim komentarzem Analityka. Obserwacja ta związana jest bezpośrednio z możliwymi przejściami między stanami zgłoszenia zaimplementowanych w JIRA dla zgłoszeń błędów projektu komercyjnego, w którym ze stanu *NInf* możliwe jest tylko przejście do *IPR*. Jeżeli takich przypadków jest dużo dobrym wyjściem optymalizacji procesu byłoby dodanie przejścia stanu z *NInf* do *V/C* bądź *R_Rej* (status ten jednak został usunięty z przepływu zgłoszeń). Opracowane przeze mnie skrypty w szczególności agregacja ścieżek kompatybilnych umożliwiają ocenę liczby zgłoszeń przechodzących przez status *NInf* , co w połączeniu z grafem IHG pomoże w podjęciu decyzji optymalizacyjnych procesu.

Istotnym profilem zaangażowania aktorów w proces obsługi zgłoszeń jest powiązanie ich aktywności z konkretnymi typami zgłoszeń. W projekcie komercyjnym P1 widoczny jest jasny podział ról zespołu, poprzez ograniczenie zaangażowania w obsługę określonych typów zgłoszeń, np. zgłoszenie typu *New Feature* procesowane są przy udziale profili aktorów: Właściciel Produktu oraz Analityk. Zgłoszenia typu *User Story, Task* – Analityk, Programista, Tester. Zgłoszenia błędów (*Bug*) – Tester, Programista, Analityk. Profile aktorów posortowane są względem poziomu zaangażowania w obsługę zgłoszeń poszczególnych typów. Poziom zaangażowania odzwierciedla się także w liczbie obsługiwanych stanów przez profile aktorów.

Analiza zaangażowanych grup aktorów w projektach *open source* jest utrudniona z uwagi na brak jasnego podziału uczestników procesu. Narzędzie JIRA udostępnia możliwość organizacji użytkowników narzędzia w grupy/role, które ułatwiałyby taką analizę. Jednak informacja ta nie jest utrzymywana w projektach *open source* lub dostęp do niej jest ograniczony wyłącznie dla uczestników projektu poprzez konfigurację uprawnień.

Role użytkowników można by określić za pomocą klasyfikacji (*machine learning*), przykład takiej klasyfikacji przedstawiłem w sekcji 4.4.

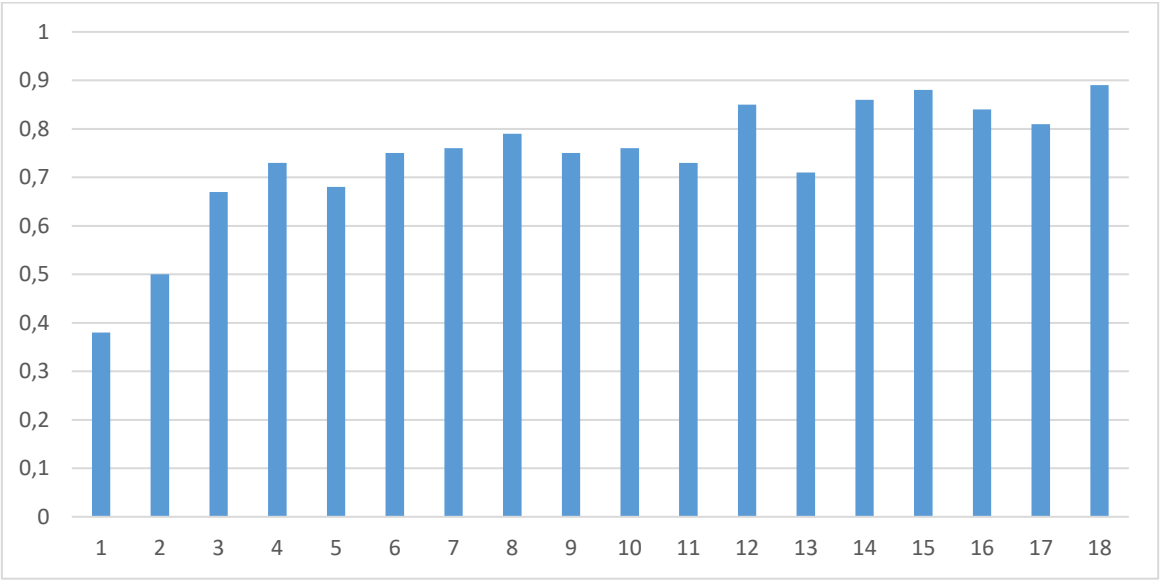
W analizie korelacji zgłoszeń oraz zmian kodu warto prześledzić obciążenie aktorów projektów realizujących te zmiany. Zaproponowałem tu miarę PRZ zdefiniowaną poniżej:

$$PRZ = \sum_{i \in nz} \frac{lp_i / lz_i}{nz}$$

Gdzie: *nz* – liczba analizowanych zgłoszeń, które wymagały zmian kodu, lp_i oraz lz_i – liczba programistów zaangażowanych w zmiany kodu oraz liczba zmian kodu dotyczących i-tego zgłoszenia. Profil *PRZ* można odnieść do zdefiniowanego zbioru zgłoszeń, np. dotyczącego określonego typu, priorytetu. Niska wartość *PRZ* może wskazywać na:

- Niepoprawną granulację zadań w projekcie
- Problemy w opisach zgłoszeń, niejasność zadań
- Problemy z dostarczaniem rozwiązania, np. brak przeprowadzonych testów przez programistę w środowisku lokalnym przed przekazaniem zmiany do testów

Dla ilustracji wyznaczyłem profile *PRZ* dla projektu P1 w okresie 18 kolejnych miesięcy (Rys. 10).



Rys. 10 Histogram profilu *PRZ* projektu P1, w okresie 18 miesięcy

W projekcie komercyjnym P1 współczynnik dąży do wartości 1. Dzięki możliwości weryfikacji treści zgłoszeń w repozytorium błędów na przestrzeni całego projektu, poprawne okazały się również założenia, które określone zostały dla niskich wartości *PRZ*.

W początkowych fazach projektu zespół raportował kilka błędów w jednym zgłoszeniu. Jedno zgłoszenie błędu określało dwa oddzielne błędy, które mogły być rozwiązane przez dwie różne osoby. Wraz ze wzrostem znajomości projektu, przez zespół oraz wypracowania optymalnego sposobu opisywania zgłoszeń błędów, wartości profilu *PRZ* rosła, co potwierdza przyjęte założenia, pozwalające na ocenę złożoności błędów oraz procesu rozwiązania zgłoszenia. Metryka ta może być wyliczona także dla projektów *open source*.

6. Podsumowanie

Tematyka badań obejmuje wielowymiarową analizą i eksplorację procesu obsługi zgłoszeń i jest wynikiem moich praktycznych doświadczeń z rozwijaniem i monitorowaniem projektów programowych. Pogłębione studia literaturowe zaowocowały sformułowaniem tezy pracy, którą przedstawiono w rozdziale 2. Przeprowadzone badania dotyczą dwóch powiązanych ze sobą kierunków:

1) Eksploracji zawartości informacyjnej programowych repozytoriów zgłoszeń
2) Wielowymiarowej analizy obsługi zgłoszeń

Badania łączyły w sobie aspekty teoretyczne oraz praktyczne. Wymagały one opracowania szeregu narzędzi wspomagających prace eksperymentalne. Do najważniejszych osiągnięć pracy należy:

W ramach tematu 1:

- Opracowanie profili statystycznych zawartości repozytoriów
- Opracowanie profili aktywności aktorów (uczestników) projektu
- Wielowymiarowa eksploracja opisów oraz komentarzy zgłoszeń

W ramach tematu 2:

- Zdefiniowanie ogólnych i szczegółowych charakterystyk procesu obsługi zgłoszeń
- Opracowanie grafowego modelu obsługi zgłoszeń (IHG) obejmującego różne perspektywy obserwacji
- Opracowanie algorytmów analizy schematów obsługi zgłoszeń w kontekście zaproponowanej ich taksonomii
- Przeprowadzenie analiz reprezentatywnego zbioru repozytoriów programowych przy wykorzystaniu opracowanych narzędzi programowych

Profile statystyczne, które przedstawiłem w rozdziale 3., dotyczą różnych atrybutów zgłoszeń. Rozpatrzyłem tu również pewne własności korelacyjne (sekcja 3.3.4). Prowadzone dyskusje potwierdzają zasadność przeprowadzania drobnoziarnistych analiz dostępnych danych, które mogą ujawniać niedoskonałości raportowania projektu. W szczególności dotyczące wypełniania pól zgłoszeń zaprezentowanych w sekcji 3.3.1., poprawności przypisywania odpowiednich wartości atrybutów, jednoznaczności skali tych wartości w cyklu życia projektu itp.

Ocenę aktywności aktorów (sekcja 3.3.2) oparłem o profile z różnymi agregacjami dystrybucji tworzonych zgłoszeń. Zauważalna była tendencja do tworzenia się grup

użytkowników odpowiadających za większość dodawanych zgłoszeń, np. dla projektu Apache Casandra 80% raportów zostało dodanych przez 14,7% reporterów, a w projekcie komercyjnym 9 testerów odpowiadało za dodanie 64,89% zgłoszeń. Profil aktywności aktorów pozwala wyróżnić uczestników o mniejszym udziale w projekcie, jak i najbardziej aktywnych członków zespołu. Informacja ta jest szczególnie wartościowa w analizach projektów komercyjnych i może zostać wykorzystana do kalibracji prac zespołu. Agregacje tak szczegółowych danych są zazwyczaj pomijane w standardowych podejściach, jednak w praktycznym doświadczeniu wykazały dużą przydatność przy ocenie zaangażowania poszczególnych aktorów projektu komercyjnego. Przedstawione charakterystyki uwzględniają również profile dodawanych komentarzy zgłoszeń, które dają wgląd w zaangażowanie aktorów w kolejne etapy obsługi zgłoszenia, jak np. dyskusja na temat rozwiązania zgłoszenia. Szczegółową analizę komentarzy mogłem przeprowadzić jedynie w przypadku projektu komercyjnego. Dla projektów *open source* zaobserwowałem brak określania roli aktora w JIRA, co wskazuje na brak odpowiedniej konfiguracji narzędzia.

W ramach prowadzonej eksploracji tekstów (rozdział 4.), przeprowadziłem szczegółową analizę słów występujących w opisach zgłoszeń. Wartościowym aspektem analiz jest przeprowadzenie podziału słów na różne słowniki: NLW (słowa języka naturalnego), FW (określenia projektowe np. nazwy własne, frazy, wycinki kodu, itp.), CCW (frazy zgodne z notacją *CamelCase*), NCW (słowa niesklasyfikowane) .Słownik FW budowany jest w oparciu o opracowane wyrażenia regularne, które zaimplementowałem na podstawie przeprowadzonych obserwacji dostępnych treści. Opisy zgłoszeń składają się w znacznym procencie ze słów języka naturalnego, które przemieszane są w różnym stopniu ze słowami spoza NLW (np. nazwy klas, modułów kodu, odsyłaczy itp.). Zastąpienie takich słów dedykowanymi tagami pozwoliło na usprawnienie dalszych eksploracji tekstu. Opracowany algorytm klasyfikacji (Alg. 4) pozwolił na przebadanie wpływu różnych konfiguracji klasyfikatorów oraz cech danych wejściowych na dokładność klasyfikacji. Problem ten został zweryfikowany dla dwóch zadań klasyfikacji. Uzyskane wyniki potwierdzają pozytywny wpływ opracowanych cech tekstów wykorzystanych w mechanizmie na precyzję wyniku. Zadanie klasyfikacji typu zgłoszenia nie było moim głównym celem. Jednak, udało mi się wskazać, że precyzja klasyfikacji na podstawie dodatkowych klas może być pewnym wyznacznikiem jakości opisów zgłoszeń w repozytorium oraz wpływa pozytywnie na dokładność klasyfikacji kategorii komentarzy. Opracowana metodyka może być wykorzystana do innych zadań klasyfikacji np. wskazanych w rozdziale 4.4.

Znacząca część pracy dotyczy metodyki wielowymiarowej analizy obsługi zgłoszeń bazujących na wprowadzonym modelu IHG (rozdział 5.) oraz zestawie oryginalnych algorytmów (Alg. 6, Alg. 7, Alg. 9, Alg. 12). Metodyka ta umożliwia przeprowadzenie dokładnej ewaluacji procesu obsługi zgłoszeń w różnych perspektywach, np. ogólnej – uwzględniającej wszystkie typy zgłoszeń, lub szczegółowej – poprzez stosowanie różnych ograniczeń R, takich jak typ zgłoszeń, grupy reporterów, priorytet zgłoszeń itp. Analiza różnych perspektyw pozwala na przeprowadzenie porównań, np. pomiędzy procesami obsługi różnych priorytetów zgłoszeń, typów zgłoszeń lub różnych projektów. W przeprowadzaniu porównań wartościowe są opracowane profile czasowe i strukturalne określające wydajność stanów i ścieżek. Model IHG oraz oparte o niego analizy wspierane są przez szereg algorytmów, które dotyczą pobierania zgłoszeń oraz ich przetworzenia do opracowanego modelu ścieżek (Alg. 6), agregacji przetworzonych danych do modelu IHG wraz z wyliczeniem profili wydajnościowych, czasowych i strukturalnych (Alg. 7), filtracji grafu (Alg. 8, Alg. 9), wyznaczenia anomalii (Alg. 12). Implementacje algorytmów tworzą narzędzie *IssueAnalyzerTool*. Opracowana metodyka została wykorzystana do przebadania szeregu repozytoriów projektów *open source* oraz jednego komercyjnego. W tym ostatnim przypadku miałem większe możliwość interpretacji wyników.

W ramach prac badawczych określiłem również algorytm weryfikacji ścieżek kompatybilnych opartych o zadane wzorce (sekcja 5.2.5). Na szczególną uwagę zasługują ścieżki agregowane przez wzorce regularne, które wspomagają szeroką perspektywę analiz anomalii. Pokazują one powiązane z nimi skumulowane profile strukturalne i czasowe, które pozwalają na ocenę wpływu pętli stanów na proces obsługi zgłoszeń. Wartościowe są również badania problemu długu błędów (rozdział 5.1). Opracowana analiza długu błędów różni się od innych propozycji z literatury. Przykładowo w pracy [107] przedstawione w kwietniu na konferencji ENASE 2023 dyskutowany jest „dług nie-techniczny” (ang. *Non-Technical Debt*), który może wpływać negatywnie na powodzenie projektu. Autorzy tej pracy skupiają się na ryzyku wynikającym z zaniedbań związanych z: procesem, kulturą organizacji, ludźmi, społecznością i organizacji. Ryzyka tego typu są odmienne od mojej definicji długu błędów, który adresuje ryzyko związane ze zgłoszeniami błędów zakwalifikowanych jako mało istotne i odłożone na późniejszy (nie określony) termin ich rozwiązania.

Opracowany Modeli IHG pozwala określić „wąskie gardła” (ang. *bottlenecks*), anomalie lub pomijalne aspekty (np. stany o małym pokryciu) procesu obsługi zgłoszeń. Pozyskana w ten sposób wiedza o procesie może zostać wykorzystana do optymalizacji prac

związanych z obsługą zgłoszeń oraz poprawą konfiguracji systemu raportowania zgłoszeń (ITS). Jest to krytyczny aspekt prac projektowych, a ewaluacja procesu powinna być wykonywana cyklicznie. Monitorowanie zmienności modelu IHG znacząco poprawia zrozumienie procesu obsługi zgłoszeń i umożliwia jego kalibrację oraz ocenę wprowadzanych modyfikacji w kolejnej iteracji. Drobnoziarnista analiza jest wartościowa przy optymalizacji prac, a zastosowanie modeli IHG dostarcza charakterystyki tego procesu, bez których wykrycie specyficznych anomalii byłoby trudne bądź niemożliwe.

Modele analiz przedstawione w rozprawie zostały przeze mnie wykorzystane do ewaluacji procesu obsługi zgłoszeń w projekcie komercyjnym oraz przeprowadzanie w nim szeregu usprawnień takich jak:

- Optymalizacja złożoności zgłoszeń – problem został odkryty na podstawie profilu *PRZ* (sekcja 5.2.7), a jego wyniki posłużyły do prowadzenia wewnętrznych dyskusji z zespołem co poprawiło sposób raportowania zgłoszeń
- Optymalizacja procesu obsługi zgłoszeń:
    - Analiza poszczególnych grafów IHG (pełne i szczegółowe) pozwoliły na identyfikację wąskich gardeł procesu takich jak:
        - Mało używane stany oraz przejścia między stanami
        - Stany z wysokim czasem obsługi
        - Ścieżki z wieloma stanami, wysokim czasem obsługi, dużą liczbą komentarzy
        - Pętle stanów ścieżek
    - Na podstawie identyfikowanych anomalii wykonana została: kalibracja narzędzia JIRA, analiza pojedynczych zgłoszeń (z długimi ścieżkami i czasami obsługi) oraz dyskusja zidentyfikowanych problemów obsługi zgłoszeń z członkami zespołu projektowego
    - Wprowadzone zmiany procesu obsługi zgłoszeń (konfiguracji systemu ITS i prac zespołu) podlegały ocenie dzięki możliwości generowania modelu IHG w różnych perspektywach czasowych oraz opracowanym schematom ich porównań

Szczególnie przydatne w przeprowadzonych analizach są generowane pliki *excel* (przykład w sekcji 8.3), które umożliwiają sortowanie dostępnych kolumn i szybką identyfikację niewłaściwych wartości profili. Przy analizie zgłoszeń z dużymi czasami obsługi oraz dużej liczby komentarzy wykorzystałem opracowany klasyfikator komentarzy,

który pomógł w ocenie kategorii dodawanych komentarzy oraz wskazał na pewne anomalie procesu jak np. pytania bez odpowiedzi bądź pytania dotyczące treści zgłoszenia. Przeprowadzenie takiej analizy ręcznie było by czasochłonne dla dużej liczby zgłoszeń. Metoda ta ułatwiła szybką identyfikację wspomnianych anomalii oraz stanowiło podstawę do dyskusji z zespołem projektowym. Wszystkie obserwacje zyskały pozytywną aprobatę zespołu, a wprowadzone usprawnienia pozwoliły na optymalizację procesu obsługi zgłoszeń, poprawienie kooperacji w zespole, polepszenie jakości opisów zgłoszeń a finalnie także na zwiększenie produkcyjności zespołu projektowego. Ewaluacja opracowanej metodyki przedstawionej w rozprawie na projekcie komercyjnym potwierdza jej wysoką użyteczność. Projekt ten był realizowany w nowej technologii *SCRUM*, której specyfikę uwzględniłem między innymi w analizach czasowych (analiza ścieżek przekraczających granicę sprintu). Problem ten rozpatrywałem również w publikacji [11] dotyczącej innego projektu komercyjnego. Zagadnienie to było pomijane w literaturze.

Dalszy kierunek badań w obszarze procesu obsługi zgłoszeń mógłby skupić się na poszerzeniu analiz oraz źródeł danych o kolejne repozytoria wiedzy, np. GitHub oraz przebadaniu korelacji ze zmianami kodu. W przypadku dysponowania repozytoriami projektów komercyjnych oraz współpracy z aktorami projektu, wyniki analiz można by konfrontować z ich opiniami. Obiecującym kierunkiem badań jest włączenie aktorów do etykietowania zgłoszeń dla różnych kategorii klasyfikacji (np. ocena wartości diagnostycznych opisów). Możliwe jest także rozbudowanie modelu IHG o powiązanie z aktywnościami aktorów w poszczególnych stanach grafu, np. którzy aktorzy zadają pytania i w jakim stanie, kto na nie odpowiada i czy jest to skorelowane ze stanem zgłoszenia. Identyfikacja aktorów niefrasobliwie nadających wartość atrybutów zgłoszeń (priorytet) lub powodujących anomalie w ich obsłudze, itp. Dodatkowo, możliwe jest rozwinięcie analiz tekstowych oraz wzbogacenie narzędzi o sztuczną inteligencję dokonującą predykcji lub sugestii zmian w tle działających systemów, wyodrębnienie powodów anomalii (np. pętli w obsłudze błędów), niepoprawnego przypisania aktorów do rozwiązania zgłoszeń, itp. Możliwe jest też uwzględnienie innych technik analizy tekstu z literatury np. opisanych w ostatnio opublikowanej pracy [108]. Praca ta poświęcona jest badaniom sentymentów komentarzy z wykorzystaniem uczenia maszynowego oraz modelu LSTM. Zaprezentowane podejście mogłoby zostać wykorzystane jako usprawnienie wyznaczania cechy sentymentu komentarzy, które w rozprawie wykorzystuje gotowe moduły biblioteki NLTK. Planowane są także dalsze publikacje na podstawie wyników analiz prowadzonych w czasie przygotowywania rozprawy.

7. Bibliografia

1. Git Basics Episode 2. What is Git? [Online]. Available from: https://git-scm.com/video/what-is-git. [Dostęp 12.03.2023].
2. Subversion system [Online]. Available from: https://subversion.apache.org/features.html. [Dostęp 12.03.2023].
3. Google Trends [Online]. Available from: https://trends.google.com/trends/explore?date=all&q=git,svn&hl=pl. [Dostęp 16.03.2023].
4. JIRA Software [Online]. Available from: https://www.atlassian.com/pl/software/jira/features. [Dostęp 12.03.2023].
5. Bugzilla [Online]. Available from: https://www.bugzilla.org/. [Dostęp 12.03.2023].
6. GitHub Issues [Online]. Available from: https://docs.github.com/en/issues/tracking-your-work-with-issues/quickstart. [Dostęp 13.03.2023].
7. Redmine [Online]. Available from: https://www.redmine.org/. [Dostęp 13.03.2023].
8. OTRS Software [Online]. Available from: https://otrs.com/product-otrs/. [Dostęp 13.03.2023].
9. Mantis Bug Tracker [Online]. Available from: https://www.mantisbt.org/. [Dostęp 13.03.2023].
10. Dobrzyński, B. , Sosnowski, J. Tracing life cycle of software bugs. in International Conference on Dependability and Complex Systems. 2016. Springer.
11. Dobrzyński, B. , Sosnowski, J. Tracing project development in Scrum model. in Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018. 2018. International Society for Optics and Photonics.
12. Dobrzyński, B., Analiza raportów błędów programowych, Praca Magisterska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych 2015.
13. Janczarek, P., Analiza efektywności obsługi błędów w procesach wytwarzania i eksploatacji oprogramowania, Praca Doktorska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych. 2016.
14. Sosnowski, J., Dobrzyński, B., Janczarek, P., Analysing problem handling schemes in software projects. Information and Software Technology, 2017. 91: 56-71.
15. Cinque, M., et al. On the impact of debugging on software reliability growth analysis: a case study. in International Conference on Computational Science and Its Applications. 2014. Springer.
16. Santana, F., et al. Xflow: An extensible tool for empirical analysis of software systems evolution. in Proceedings of the VIII Experimental Software Engineering Latin American Workshop, ESELAW. 2011.
17. Mesquida, A.L. , Mas, A., A project management improvement program according to ISO/IEC 29110 and PMBOK®. Journal of Software: Evolution and Process, 2014. 26(9): 846-854.

18. Ogasawara, H., Kusanagi, T., Aizawa, M., Proposal and practice of software process improvement framework–Toshiba's software process improvement history since 2000. Journal of Software: Evolution and Process, 2014. 26(5): 521-529.
19. Sommerville, I., Software Engineering 9th Edition, vol. 137035152 ISBN-10, 2011.
20. Sosnowski, J. , Sabak, J. Software reliability analysis in designing data base oriented applications. in Proceedings 27th EUROMICRO Conference. 2001: A Net Odyssey. 2001. IEEE.
21. Natarajan, R. Top 10 Open Source Bug Tracking Systems [Online]. Available from: http://www.thegeekstuff.com/2010/08/bug-tracking-system/. [Dostęp 15.03.2023].
22. Amasaki, S., et al., Constructing a Bayesian belief network to predict final quality in embedded system development. IEICE Transactions on Information and Systems, 2005. 88(6): 1134-1141.
23. Borg, M. , Runeson, P., Changes, evolution, and bugs, in Recommendation systems in software engineering. 2014, Springer. p. 477-509.
24. Bettenburg, N., et al. What makes a good bug report? in Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering. 2008.
25. Zimmermann, T., et al., What makes a good bug report? IEEE Transactions on Software Engineering, 2010. 36(5): 618-643.
26. Garousi, V., Ergezer, E.G., Herkiloğlu, K. Usage, usefulness and quality of defect reports: an industrial case study. in Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering. 2016.
27. Fan, Y., et al., Chaff from the wheat: Characterizing and determining valid bug reports. IEEE Transactions on Software Engineering, 2018. 46(5): 495-525.
28. Aljedaani, W. , Javed, Y. Bug Reports Evolution in Open Source Systems. in 5th International Symposium on Data Mining Applications. 2018. Springer.
29. Alenezi, M., Magel, K., Banitaan, S., Efficient Bug Triaging Using Text Mining. Proceedings of the 5th International Conference on Computer Engineering and Technology, 2013.
30. Zhang, T., et al. Predicting severity of bug report by mining bug repository with concept profile. in Proceedings of the 30th Annual ACM Symposium on Applied Computing. 2015.
31. Dal Sassc, T. , Lanza, M. A closer look at bugs. in 2013 First IEEE Working Conference on Software Visualization (VISSOFT). 2013.
32. Halverson, C.A., et al. Designing task visualizations to support the coordination of work in software development. in Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work. 2006.

33. Hammad, M., Abufakher, S., Hammad, M., A visualization approach for bug reports in software systems. International Journal of Software Engineering and Its Applications, 2014. 8(10): 37-46.
34. D'Ambros, M., Lanza, M., Pinzger, M. " A Bug's Life" Visualizing a Bug Database. in 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis. 2007. IEEE.
35. Jeong, G., Kim, S., Zimmermann, T. Improving bug triage with bug tossing graphs. in Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering. 2009.
36. Kubacki, M. , Sosnowski, J. Multidimensional log analysis. in 2016 12th European Dependable Computing Conference (EDCC). 2016. IEEE.
37. Sosnowski, J., et al., Analyzing logs of the university data repository, in Intelligent Tools for Building a Scientific Information Platform: From Research to Implementation. 2014, Springer. p. 141-156.
38. Polaczek, J. , Sosnowski, J., Exploring the software repositories of embedded systems: An industrial experience. Information and Software Technology, 2021. 131: 106489.
39. Janczarek, P. , Sosnowski, J., Managing complex software projects. Information Systems in Management, 2015. 4(3): 171-182.
40. Dobrzyński, B. , Sosnowski, J. Text Mining Studies of Software Repository Contents (przyjęte do publikacji). in 18th International Conference on Evaluation of Novel Approaches to Software Engineering. 2023.
41. Garousi, V. Evidence-based insights about issue management processes: an exploratory study. in Trustworthy Software Development Processes: International Conference on Software Process, ICSP 2009 Vancouver, Canada, May 16-17, 2009 Proceedings. 2009. Springer.
42. Nayrolles, M. , Hamou-Lhadj, A. Towards a classification of bugs to facilitate software maintainability tasks. in Proceedings of the 1st International Workshop on Software Qualities and their Dependencies. 2018.
43. Izadi, M., Akbari, K., Heydarnoori, A., Predicting the objective and priority of issue reports in software repositories. Empirical Software Engineering, 2022. 27(50).
44. Jahanshahi, H., Cevik, M., Başar, A. Predicting the number of reported bugs in a software repository. in Advances in Artificial Intelligence: 33rd Canadian Conference on Artificial Intelligence, Canadian AI 2020, Ottawa, ON, Canada, May 13–15, 2020, Proceedings 33. 2020. Springer.
45. Banerjee, S., et al., Automated triaging of very large bug repositories. Information and Software Technology, 2017. 89: 1-13.

46. Xi, S.-Q., et al., Bug triaging based on tossing sequence modeling. Journal of Computer Science and Technology, 2019. 34: 942-956.
47. Xuan, J., et al., Towards effective bug triage with software data reduction techniques. IEEE Transactions on Knowledge and Data Engineering, 2014. 27(1): 264-280.
48. Gu, H., Zhao, L., Shu, C. Analysis of duplicate issue reports for issue tracking system. in The 3rd International Conference on Data Mining and Intelligent Information Technology Applications. 2011. IEEE.
49. Nadeem, A., Sarwar, M.U., Malik, M.Z. Automatic issue classifier: A transfer learning framework for classifying issue reports. in 2021 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW). 2021. IEEE.
50. Herbold, S., Trautsch, A., Trautsch, F., On the feasibility of automated prediction of bug and non-bug issues. Empirical Software Engineering, 2020. 25: 5333-5369.
51. Hanagal, D.D. , Bhalerao, N.N., Software reliability growth models. 2021: Springer.
52. Nafreen, M., et al. Connecting software reliability growth models to software defect tracking. in 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE). 2020. IEEE.
53. Liu, Z. , Kang, R., Imperfect debugging software belief reliability growth model based on uncertain differential equation. IEEE Transactions on Reliability, 2022. 71(2): 735-746.
54. Elmishali, A. , Kalech, M., Issues-Driven features for software fault prediction. Information and Software Technology, 2023. 155: 107102.
55. Yamamoto, K., et al. Which metrics should researchers use to collect repositories: an empirical study. in 2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS). 2020. IEEE.
56. Bugayenko, Y., et al., Qualitative Clustering of Software Repositories Based on Software Metrics. IEEE Access, 2023. 11: 14716-14727.
57. Hasabnis, N. GitRank: a framework to rank GitHub repositories. in Proceedings of the 19th International Conference on Mining Software Repositories. 2022.
58. Kolovos, D., et al. Crossflow: a framework for distributed mining of software repositories. in 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR). 2019. IEEE.
59. Martens, B. , Franke, J. Identifying Agile Roles in Software Engineering Projects using Repository and Work-Tracking Data. in 2022 International Conference on Data and Software Engineering (ICoDSE). 2022. IEEE.
60. Poncin, W., Serebrenik, A., Van Den Brand, M. Process mining software repositories. in 2011 15th European Conference on Software Maintenance and Reengineering. 2011. IEEE.

61. Rana, R. , Staron, M. When do software issues and bugs get reported in large open source software project? in Proceedings of the 25th International Conference on Software Measurement (IWSM). 2015.
62. Sokol, F.Z., Aniche, M.F., Gerosa, M.A. MetricMiner: Supporting researchers in mining software repositories. in 2013 IEEE 13th International Working Conference on Source Code Analysis and Manipulation (SCAM). 2013. IEEE.
63. Singh, V., Misra, S., Sharma, M., Bug severity assessment in cross project context and identifying training candidates. Journal of Information & Knowledge Management, 2017. 16(01): 1750005.
64. Lunesu, M.I., et al., Assessing the Risk of Software Development in Agile Methodologies Using Simulation. IEEE Access, 2021. 9: 134240-134258.
65. Yadav, A., Singh, S.K., Suri, J.S., Ranking of software developers based on expertise score for bug triaging. Information and Software Technology, 2019. 112: 1-17.
66. Hussain, M., et al., Prioritizing the issues extracted for getting right people on right project in software project management from vendors' perspective. IEEE Access, 2021. 9: 8718-8732.
67. Jiang, Y., et al., LTRWES: A new framework for security bug report detection. Information and Software Technology, 2020. 124: 106314.
68. Peters, F., et al., Text filtering and ranking for security bug report prediction. IEEE Transactions on Software Engineering, 2017. 45(6): 615-631.
69. Panichella, S., Canfora, G., Di Sorbo, A., "Won't We Fix this Issue?" Qualitative characterization and automated identification of wontfix issues on GitHub. Information and Software Technology, 2021. 139: 106665.
70. Wu, X., et al., Invalid bug reports complicate the software aging situation. Software Quality Journal, 2020. 28: 195-220.
71. Wang, H. , Yuan, L., Software engineering defect detection and classification system based on artificial intelligence. Nonlinear Engineering, 2022. 11(1): 380-386.
72. Umer, Q., Liu, H., Sultan, Y., Sentiment based approval prediction for enhancement reports. Journal of Systems and Software, 2019. 155: 57-69.
73. Ramirez-Mora, S.L., et al., Exploring the communication functions of comments during bug fixing in Open Source Software projects. Information and Software Technology, 2021. 136(106584).
74. Arya, D., et al. Analysis and detection of information types of open source software issue discussions. in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE). 2019. IEEE.
75. Huang, Y., et al., An empirical study on the issue reports with questions raised during the issue resolving process. Empirical Software Engineering, 2019. 24: 718-750.

76. Choetkiertikul, M., et al., Automatically recommending components for issue reports using deep learning. Empirical Software Engineering, 2021. 26: 1-39.
77. Elmishali, A., et al. Beirut: Repository mining for defect prediction. in 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE). 2021. IEEE.
78. Aljedaani, W., Javed, Y., Alenezi, M. Open source systems bug reports: meta-analysis. in Proceedings of the 2020 The 3rd International Conference on Big Data and Education. 2020.
79. Rath, M. , Mäder, P., Structured information in bug report descriptions—influence on IR-based bug localization and developers. Software Quality Journal, 2019. 27: 1315-1337.
80. Gomes, L.A.F., da Silva Torres, R., Côrtes, M.L., Bug report severity level prediction in open source software: A survey and research opportunities. Information and Software Technology, 2019. 115: 58-78.
81. Qamar, K.A., Sülün, E., Tüzün, E., Taxonomy of bug tracking process smells: Perceptions of practitioners and an empirical analysis. Information and Software Technology, 2022. 150: 106972.
82. Polaczek, J.J., Analiza rozwoju i utrzymania oprogramowania w systemach wbudowanych, Praca Magisterska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych. 2019.
83. Lasynskyi, M. , Sosnowski, J., Extending the space of software test monitoring: practical experience. IEEE Access, 2021. 9: 166166-166183.
84. Reszka, Ł., Analiza repozytoriów zgłoszeń w cyklu życia programów, Praca Magisterska, Politechnika Warszawska, Wydział Elektroniki i Technik Informacyjnych. 2022.
85. Li, Q., et al., A survey on text classification: From traditional to deep learning. ACM Transactions on Intelligent Systems and Technology (TIST), 2022. 13(2): 1-41.
86. Gaikwad, S.V., Chaugule, A., Patil, P., Text mining methods and techniques. International Journal of Computer Applications, 2014. 85(17).
87. Rose, S., et al., Automatic keyword extraction from individual documents. Text mining: Applications and Theory, 2010: 1-20.
88. Zhang, K., et al. Keyword extraction using support vector machine. in Advances in Web-Age Information Management: 7th International Conference, WAIM 2006, Hong Kong, China, June 17-19, 2006. Proceedings 7. 2006. Springer.
89. Gegick, M., Rotella, P., Xie, T. Identifying security bug reports via text mining: An industrial case study. in 2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010). 2010. IEEE.
90. Valdez, A., et al. Sentiment analysis in Jira Software Repositories. in 2020 8th International Conference in Software Engineering Research and Innovation (CONISOFT). 2020. IEEE.

91. Zhang, W., et al., FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Information and Software Technology, 2019. 110: 121-135.
92. Zhang, T., et al., Towards more accurate severity prediction and fixer recommendation of software bugs. Journal of Systems and Software, 2016. 117: 166-184.
93. Sureka, A. , Jalote, P. Detecting duplicate bug report using character n-gram-based features. in 2010 Asia Pacific Software Engineering Conference. 2010. IEEE.
94. Hindle, A. , Onuczko, C., Preventing duplicate bug reports by continuously querying bug reports. Empirical Software Engineering, 2019. 24(2): 902-936.
95. Ebrahimi, N., et al., An HMM-based approach for automatic detection and classification of duplicate bug reports. Information and Software Technology, 2019. 113: 98-109.
96. Regular expression syntax cheat sheet Available from: https://en.wikipedia.org/wiki/Regular_expression. [Dostęp 29.03.2023].
97. NLTK: Natural Language Toolkit Available from: https://www.nltk.org/. [Dostęp 29.03.2023].
98. Mallidi, R.K. , Sharma, M., Study on Agile Story Point Estimation Techniques and Challenges. International Journal of Computer Applications, 2021. 174: 9-14.
99. scikit-learn: Machine Learning in Python Available from: https://scikit-learn.org/stable/. [Dostęp 29.03.2023].
100. Erra, U., et al., Approximate TF–IDF based on topic extraction from massive message stream using the GPU. Information Sciences, 2015. 292: 143-161.
101. Jing, L.-P., Huang, H.-K., Shi, H.-B. Improved feature selection approach TFIDF in text mining. in Proceedings. International Conference on Machine Learning and Cybernetics. 2002. IEEE.
102. Muraszkiewicz, M. , Nowak, R.M., (redaktorzy pracy zbiorowej). Sztuczna inteligencja dla inżynierów. Metody ogólne. 2022: Oficyna Wydawnicza PW.
103. Aldaeej, A. , Seaman, C. From lasagna to spaghetti: A decision model to manage defect debt. in 2018 IEEE/ACM International Conference on Technical Debt (TechDebt). 2018. IEEE.
104. Cunningham, W., The WyCash portfolio management system. ACM SIGPLAN OOPS Messenger, 1992. 4(2): 29-30.
105. Akbarinasaji, S., Bener, A.B., Erdem, A. Measuring the principal of defect debt. in Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering. 2016.
106. Java Universal Network/Graph Framework [Online]. Available from: https://jung.sourceforge.net/. [Dostęp 29.03.2023].
107. Saeeda, H., Ovais Ahmad, M., Gustavsson, T. Multivocal Literature Review on Non-Technical Debt in Software Development: An Exploratory Study (przyjęte do publikacji). in 18th International Conference on Evaluation of Novel Approaches to Software Engineering. 2023.

108. Vijayvargiya, S., et al. Software Engineering Comments Sentiment Analysis Using LSTM with Various Padding Sizes (przyjęte do publikacji). in 18th International Conference on Evaluation of Novel Approaches to Software Engineering. 2023.

8. Załączniki

8.1. Profile ścieżek obsługi zgłoszeń dla projektu Groovy

N	Path	L	AVG	MAX	Q2	Q3
2445	Open,Resolved_Fixed,Closed	3	269,7	4703,1	68,2	205,8
695	Open,Closed	2	307,7	4606,4	16,6	272,6
445	Open,In Progress,Resolved_Fixed,Closed	4	525,1	4542,8	122,2	401,1
103	Open,Resolved_Duplicate,Closed	3	347,4	2740,6	62,4	369
102	Open,Resolved_Not A Bug,Closed	3	119,1	2199,9	36,5	92,3
96	Open,Resolved_Won't Fix,Closed	3	360,6	3597,5	136,5	574,1
51	Open,Resolved_Fixed,Reopened,Resolved_Fixed,Closed	5	327,6	3621,1	64,6	211,7
42	Open,Resolved_Cannot Reproduce,Closed	3	246,4	1822,5	116,4	304,2
32	Open,In Progress,Resolved_Fixed	3	1909,6	4496,6	1826,5	3065,6
32	Open,Resolved_Fixed,Closed,Reopened,Resolved_Fixed,Closed	6	1094,5	2919,4	657,3	2210,8
21	Open,Resolved_Information Provided,Closed	3	1053	3439,6	338,4	1968,5
21	Open,Resolved_Fixed,Closed,Reopened,Closed	5	309,6	986,6	134,9	515,4
19	Open,Resolved_Not A Problem,Closed	3	317,7	1585,1	218,4	458
18	Open,Closed,Reopened,Closed	4	327,3	1305	165,2	431,2
18	Open,Resolved_Incomplete,Closed	3	465,2	1636,4	262,2	720,3
15	Open,Closed,Reopened,Resolved_Fixed,Closed	5	306,4	1158,5	232,7	320,6
12	Open,Resolved_Fixed	2	1401,7	3089,1	736,5	2720
9	Open,Resolved_Fixed,Reopened,Closed	4	246,2	1817,5	57,8	87,8
6	Open,In Progress,Open,Resolved_Fixed,Closed	5	613,6	1830	87,6	1619,8
5	Open,In Progress,Closed	3	127,2	314,5	133,3	142

Tab. 31 20 ścieżek z najwyższym pokryciek zamkniętych zgłoszeń błędów projektyu Groovy (Liczba wszystkich zamkniętych zgłoszeń błędów - 4293)

N	Path	L	AVG	MAX	Q2	Q3
606	Open	1	0	0	0	0
9	Open,In Progress	2	1021,6	2978,4	296,9	1835,3
2	Open,In Progress,Open	3	72,6	103,2	72,6	103,2
4	Open,Closed,Reopened	3	895,4	1609	986,1	1428,4
1	Open,Resolved_Fixed,Closed,Reopened	4	408,9	408,9	408,9	408,9
3	Open,Resolved_Fixed,Reopened	3	1462,1	2120,2	2054,6	2087,4
1	Open,In Progress,Resolved_Fixed,Reopened	4	46,9	46,9	46,9	46,9

Tab. 32 Ścieżki otwartych zgłoszeń błędów projektu Groovy (Liczba wszystkich otwartych zgłoszeń błędów - 626)

N	Path	L	AVG	MAX	Q2	Q3
690	Open,Resolved_Fixed,Closed	3	241,1	4775,7	68,6	154,5
157	Open,Closed	2	415	3588,9	12,6	442,7
49	Open,In Progress,Resolved_Fixed,Closed	4	505,8	3562,8	78,6	297,4
28	Open,Resolved_Won't Fix,Closed	3	459,7	2241,8	125,5	726,6
21	Open,Resolved_Duplicate,Closed	3	456,9	2637,5	61,4	854,9
9	Open,In Progress,Resolved_Fixed	3	1573,4	3376,5	1811,2	2925,3
9	Open,Resolved_Fixed,Reopened,Resolved_Fixed,Closed	5	68	141,3	55,9	86,2
8	Open,Closed,Reopened,Closed	4	271,2	1812,2	8,7	168,3
7	Open,Resolved_Fixed,Closed,Reopened,Resolved_Fixed,Closed	6	876,6	3025	495,3	1108,7
5	Open,Resolved_Fixed	2	1589	2897,2	1601,9	1812,2
5	Open,Resolved_Not A Bug,Closed	3	70,1	140,6	74	88
5	Open,Closed,Reopened,Resolved_Fixed,Closed	5	1212,1	2959,3	864,2	1515
4	Open,Resolved_Fixed,Reopened,Closed	4	57,3	140,3	42,1	104,2
2	Open,Resolved_Incomplete,Closed	3	313,1	603,5	313,1	603,5
2	Open,Resolved_Implemented,Closed	3	178,9	354,5	178,9	354,5

Tab. 33 Wybrane ścieżki z najwyższym pokryciem zamkniętych zgłoszeń Improvement projektu Groovy (Liczba wszystkich zamkniętych zgłoszeń Improvement - 1025)

N	Path	L	AVG	MAX	Q2	Q3
239	Open	1	0	0	0	0
2	Open,Resolved_Fixed,Reopened	3	1	1,9	1	1,9
2	Open,In Progress,Open	3	1258,2	1646,6	1258,2	1646,6
2	Open,Closed,Reopened	3	233,5	465,9	233,5	465,9
1	Open,Closed,Reopened,Closed,Reopened	5	1494,8	1494,8	1494,8	1494,8
1	Open,Resolved_Won't Fix,Reopened	3	0,7	0,7	0,7	0,7
1	Open,Resolved_Fixed,Closed,Reopened	4	1096	1096	1096	1096

Tab. 34 Ścieżki otwartych zgłoszeń Improvement projektu Groovy (Liczba wszystkich otwartych zgłoszeń Improvement - 248)

N	Path	L	AVG	MAX	Q2	Q3
150	Open,Resolved_Fixed,Closed	3	313,3	3812,6	95,9	323,7
52	Open,Closed	2	628,3	3647,5	98,6	982,7
16	Open,Resolved_Won't Fix,Closed	3	584,8	1758,6	609,8	857,9
6	Open,Resolved_Duplicate,Closed	3	135,2	254,4	129,2	216,7
5	Open,In Progress,Resolved_Fixed,Closed	4	1050,7	3552,3	352,2	1240,3
3	Open,Resolved_Fixed,Reopened,Closed	4	62,8	136,3	26,5	81,4
3	Open,Resolved_Fixed,Closed,Reopened,Resolved_Fixed,Closed	6	2312,2	3084,7	2212,6	2648,7
3	Open,Resolved_Fixed,Reopened,Resolved_Fixed,Closed	5	36,4	67,2	37,7	52,5
2	Open,Closed,Reopened,Closed	4	92,2	104,3	92,2	104,3
1	Open,Resolved_Invalid,Closed	3	5,5	5,5	5,5	5,5
1	Open,Resolved_Not A Problem,Closed	3	33,2	33,2	33,2	33,2
1	Open,Resolved_Fixed,Closed,Reopened,Closed	5	313,1	313,1	313,1	313,1
1	Open,Resolved_Not A Bug,Reopened,Closed	4	0,4	0,4	0,4	0,4
1	Open,Resolved_Information Provided,Closed	3	4366,2	4366,2	4366,2	4366,2
1	Open,Resolved_Implemented,Closed	3	4460,3	4460,3	4460,3	4460,3

Tab. 35 Ścieżki zamkniętych zgłoszeń New Feature projektu Groovy (Liczba wszystkich zamkniętych zgłoszeń New Feature - 246)

N	Path	L	AVG	MAX	Q2	Q3
75	Open	1	0	0	0	0
1	Open,Closed,Reopened	3	2,5	2,5	2,5	2,5
1	Open,Resolved_Won't Fix,Reopened	3	13,3	13,3	13,3	13,3

Tab. 36 Ścieżki otwartych zgłoszeń (New Feature) projektu Groovy (Liczba wszystkich otwartych zgłoszeń New Feature - 77)

N	Path	L	AVG	MAX	Q2	Q3	A	A*
2	O,R_F,Reo,R_F,Reo,R_F,C	7	148,6	194,4	148,6	194,4	[[R_F, Reo, R_F]:3]	[3: 1]
2	O,C,Reo,C,Reo,C	6	731,2	839,3	731,2	839,3	[[C, Reo, C]:3]	[3: 1]
1	O,R_NB,C,Reo,C,Reo,C	7	102,9	102,9	102,9	102,9	[[C, Reo, C]:3]	[3: 1]
1	O,R_F,C,Reo,R_F,C,Reo,R_F,C	9	375,4	375,4	375,4	375,4	[[R_F, C, Reo, R_F]:4, [C, Reo, R_F, C]:4]	[4: 2]

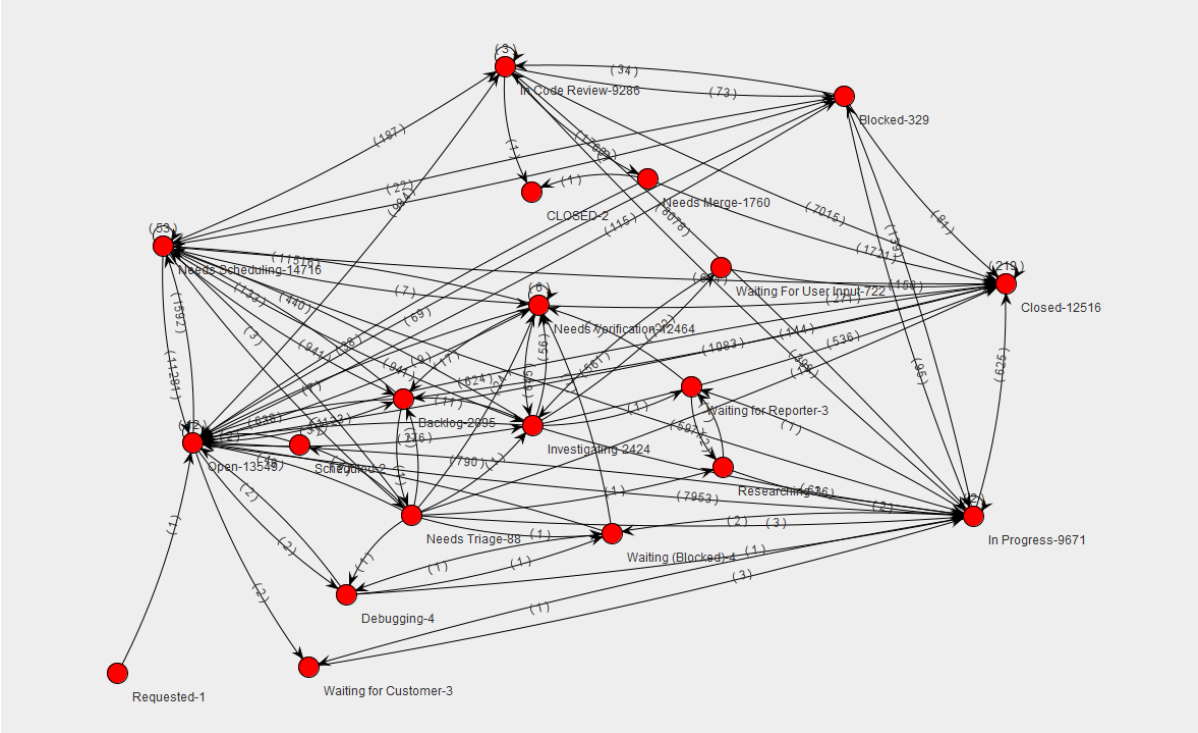
Tab. 37 Anomalie ścieżek obsługi zgłoszeń błędów projektu Groovy (Łączna liczba zgłoszeń błędów zamkniętych i otwartych 4919)

N	Path	L	AVG	MAX	Q2	Q3	A	A*	Typ	Stan T
1	O,C,Reo,C,Reo, C,Reo,R_CnR,C	9	449,6	449,6	449,6	449,6	[[C, Reo, C]:3, [Reo, C, Reo]:3]	[3: 2]	Improvement	Closed
1	O,C,Reo,C,Reo, C,Reo,C	8	874	874	874	874	[[C, Reo, C]:3, [Reo, C, Reo]:3, [C, Reo, C]:3]	[3: 3]	Improvement	Closed

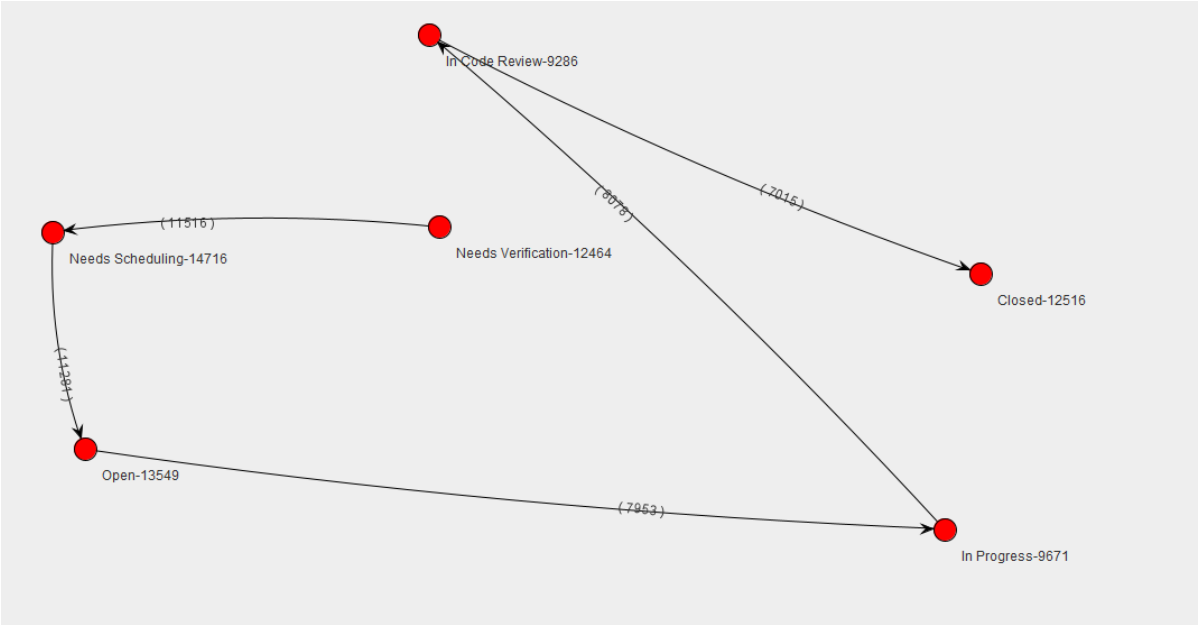
Tab. 38 Anomalie ścieżek obsługi zgłoszeń nowych funkcjonalności (typ: New Feature, Improvement; stan terminalny: Closed) projektu Groovy – łączna liczba zgłoszeń 1596

Oznaczenia kolumn: N – liczba zgłoszeń; Path – ścieżka obsługi zgłoszenia; L – liczba stanów w ścieżce obsługi zgłoszenia; AVG – średni czas obsługi ścieżki, MAX – maksymalny czas obsługi ścieżki zgłoszenia; Q2, Q3 – Kwartyl Q2,Q3 czasu obsługi ścieżki zgłoszenia; A – struktura anomalii pętrli w formie list [stany pętli: długość pętli]; A - liczba anomalii w odniesieniu do długości pętli*

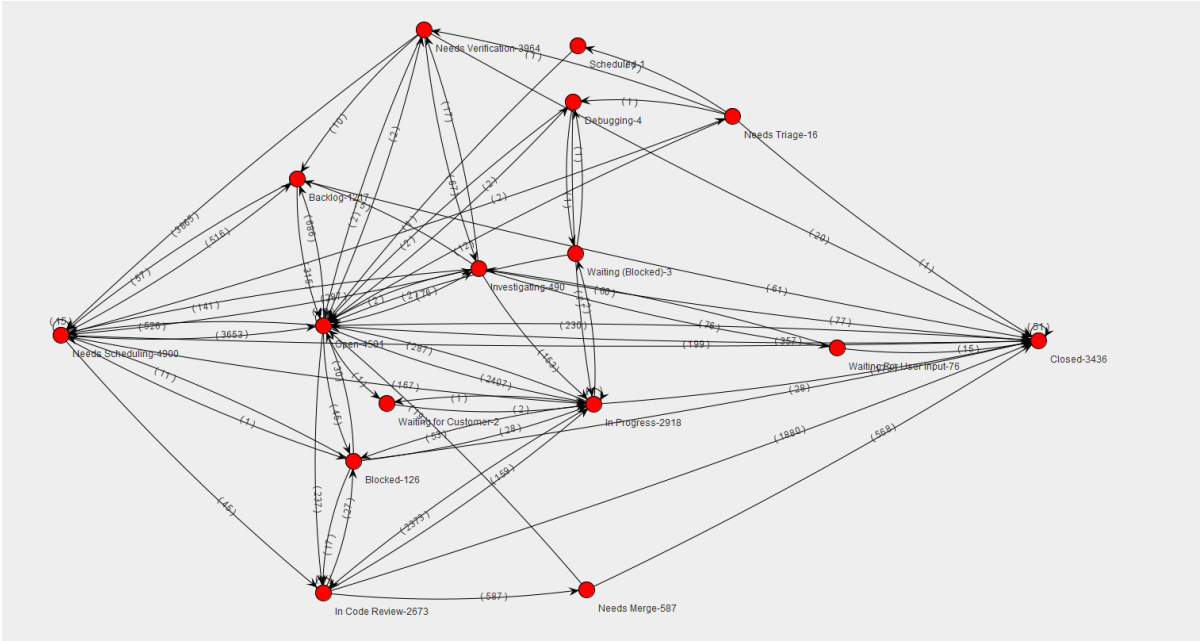
8.2. Grafy IHG dla projektu MongoDB



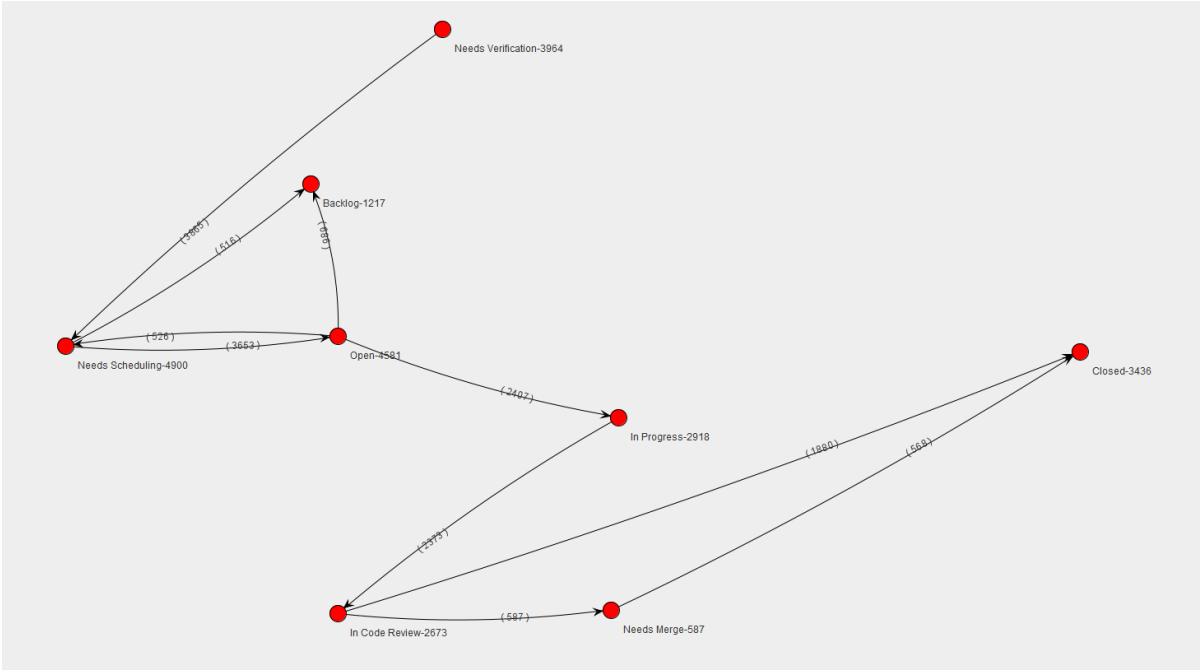
Rys. 11 Pełen graf IHG dla projektu MongoDB (łączna liczba zgłoszeń 13435: zgłoszenia błędów – 8405, zgłoszenia New Feature, Improvement– 5030)



Rys. 12 Grafł IHG dla projektu MongoDB z filtracją istotności 20% dla wszystkich zgłoszeń (łączna liczba zgłoszeń 13435: zgłoszenia błędów - 8405, zgłoszenia New Feature, Improvement - 5030)



Rys. 13 Graf IHG projektu MongoDB z R: typ – Improvement (łączna liczba zgłoszeń 4257)



Rys. 14 Graf IHG projektu MongoDB z R: typ – Improvement z filtracją istotności 10% (łączna liczba zgłoszeń 4257)

8.3. Zestaw atrybutów ścieżek obsługi zgłoszeń

Plik excel na podstawie którego prowadzone były analizy w rozdziale 5.2 składa się z zakładek opisanych w Tab. 39 oraz kolumn których opis przedstawiłem w Tab. 40.

Sufix zakładki	Opis
Bugs	Zawiera zgłoszenia błędów (stan końcowy zamknięty oraz otwarty). Kolumna (Status Procesu) definiuje czy dana ścieżka zakończyła się w stanie terminalnym - Closed (status zgłoszenia: Closed, R_,) lub czy jest nadal procesowana Open (status zgłoszenia inny niż Closed, R_,)
ClosedBugsPriority	Ścieżki zgłoszeń błędów zamkniętych, rozdzielone względem priorytetu zgłoszenia - Priorytet określa kolumna (Priorytet) z możliwymi wartościami: • Blocker • Critical • Major • Trivial, Minor Suma liczby zgłoszeń w poszczególnych wierszach (pierwsza kolumna) odpowiada sumie wartości pierwszej kolumny zakładki Bugs dla Statusu Procesu = Closed
Tasks	Ścieżki zgłoszeń typu: • Improvement, • New Feature z uwzględnieniem zgłoszeń otwartych oraz zamkniętych. Typ zgłoszenia określa przedostatnia kolumna Typ zgłoszenia. Ostania kolumna (Status Procesu) określa czy zgłoszenie osiągnęło stan terminalny (wartość Closed), czy jest nadal procesowan (wartość open).

Tab. 39 Opis zakładek pliku excel

Plik załączony został w elektronicznym załączniku pracy (pendrive), zawiera on szczegółowe charakterystyki ścieżek dla projektów (w nawiasie podana łączna liczba zgłoszeń projektu dla wszystkich zakładek):

- Groovy (6515)
- MongoDB (13435)
- Lucene Core (8685)
- Log4J2 (3108)
- Arrow (15773)

Nazwa kolumny	Opis
C	Liczba zgłoszeń przechodzących przez ścieżkę
Path	Ścieżka obsługi zgłoszenia (następujące po sobie stany)
L	Długość ścieżki określana jako liczba stanów w ścieżce
Keys	Lista identyfikatory zgłoszeń przechodzących przez ścieżkę (ograniczone do 10 pozycji, wykorzystywane jedynie w celu śledzenia anomalii w procesie obsługi zgłoszeń)
Min Time	Minimalny czas przejście zgłoszenia przez ścieżkę (od stworzenia zgłoszenia do czasu wejścia w ostatni stan ścieżki)
AVG	Średni czas obsługi ścieżki
MAX	Maksymalny czas przejście zgłoszenia przez ścieżkę
Q1	I kwartyl czasu przejścia zgłoszenia przez ścieżkę
Q2	II kwartyl czasu przejścia zgłoszenia przez ścieżkę
Q3	III kwartyl czasu przejścia zgłoszenia przez ścieżkę
Min C of comments	minimalna liczba komentarzy występująca w zgłoszeniach przechodzących przez ścieżkę
Avg C of comments	średnia liczba komentarzy występująca w zgłoszeniach przechodzących przez ścieżkę
Max C of comments	maksymalna liczba komentarzy występująca w zgłoszeniach przechodzących przez ścieżkę
Min words C in comment	minimalna liczba słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
Avg words C in comment	średnia liczba słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
Max words C in comment	maksymalna liczba słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
Min unique words C in comment	minimalna unikalnych liczba słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
Avg unique words C in comment	średnia liczba unikalnych słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
Max unique words C in comment	maksymalna liczba unikalnych słów występujących w komentarzach zgłoszeń przechodzących przez ścieżkę
A	Struktura anomalii w formie listy: [stany w pętli: długość pętli]
A*	Liczba anomalii w odniesieniu do długości pętli
Priorytet	Priorytet zgłoszeń błędów z możliwymi wartościami: • Blocker • Critical • Major • Trivial, Minor Kolumna występuje tylko w zakładkach z sufixem bugs
Type	Typ zgłoszenia z możliwymi wartościami: • Improvement • New Feature
Status procesu	Stan proces ścieżki, z możliwymi wartościami: • Open (ścieżka bez stanu terminalnego jako ostatni ustawiony stan) • Closed (ścieżka z ustawionym stanem terminalnym jako ostatni) Kolumna nie występuje w zakładkach z sufixem ClosedBugsPriority

Tab. 40 Opis kolumn pliku excel (data.xlsx)

8.4. Szczegółowe charakterystyki ścieżek obsługi zgłoszeń

N	Path	L	Keys	Min Time	AVG	MAX	Q1	Q2	Q3	Min C of comments	Avg C of comments	Max C of comments
9	NV,NSch,O, IPrs,ICR,C,C	7	SERVER-67922,	20,2	134,7	344	35,5	80	228,9	0	3	9
4	NV,NSch,O, IPrs,ICR,NM,C,C	8	SERVER-68432,SERVER-59422	6	73,3	219,8	6	33,6	131,5	1	1	2
3	NV,NSch,O, IPrs,ICR,IPrs,ICR,IPrs,ICR,C	10	SERVER-70082,SERVER-57697,SERVER-52631	34,9	61,7	99,3	34,9	50,8	75,1	2	2	3
3	NV,NSch,O, IPrs,O,IPrs,O,IPrs,ICR,C	10	SERVER-68314,SERVER-47620,SERVER-46367	49,9	178,7	357	49,9	129,2	243,1	2	2	2
3	NSch,O,NSch, NSch,O,IPrs,ICR,C	8	SERVER-55227,SERVER-47881,SERVER-46661	0,3	42,8	124,8	0,3	3,4	64,1	1	1	3
3	NV,NSch,O,O,IPrs,ICR,C	7	SERVER-49228,SERVER-47659,SERVER-47657	250,1	310,3	364,2	250,1	316,6	340,4	1	1	1
3	NV,NSch,O,IPrs,ICR,C,C	7	SERVER-68847,SERVER-60967,SERVER-45491	43,1	105,7	188,8	43,1	85,1	137	1	2	4
2	NV,NSch,O, IPrs,ICR,C,O,C,O,C	10	SERVER-71473,SERVER-56840	5,8	21,1	36,4	5,8	21,1	36,4	4	6	8
2	NV,NSch,O, IPrs,O,IPrs,ICR,C,C	9	SERVER-70212,SERVER-58476	44,2	92	139,9	44,2	92	139,9	1	2	4
2	NV,NSch,O,IPrs,C,C	6	SERVER-68999,SERVER-66752	14,1	57,1	100,1	14,1	57,1	100,1	0	3	7
2	NV,NSch ,Back,O,IPrs,ICR,C,C	8	SERVER-66956,SERVER-66692	37	108,7	180,3	37	108,7	180,3	1	2	4

Tab. 41 Wybrane wiersze pliku excel cz.1

Tab. 41 oraz Tab. 42, przedstawiają wybrane wiersze pliku *excel* (data.xlsx), którego pełna wersja dostępna jest w elektronicznym załączniku pracy (pendrive). Dla zwiększenia czytelności tabela podzielona została na dwie części, obydwie zawierają te same wiersze z pliku excel jednak w ujęciu różnych kolumn.

Min words C in comment	Avg words C in comment	Max words C in comment	Min unique words C in comment	Avg unique words C in comment	Max unique words C in comment	A	A*	Typ	Status procesu
0	30	146	0	26	99	[[C]:1]	[1: 1]	Improvement	Closed
6	26	50	6	25	47	[[C]:1]	[1: 1]	Improvement	Closed
27	30	33	24	27	32	[[IPrs, ICR, IPrs]:3, [ICR, IPrs, ICR]:3]	[3: 2]	Improvement	Closed
21	25	32	17	23	30	[[O, IPrs, O]:3, [IPrs, O, IPrs]:3]	[3: 2]	Improvement	Closed
28	33	60	26	29	43	[[NSch]:1]	[1: 1]	Improvement	Closed
31	33	38	28	30	35	[[O]:1]	[1: 1]	Improvement	Closed
2	26	36	2	24	34	[[C]:1]	[1: 1]	New Feature	Closed
24	30	36	22	27	33	[[C, O, C]:3]	[3: 1]	Improvement	Closed
7	28	39	7	27	36	[[C]:1]	[1: 1]	Improvement	Closed
0	37	178	0	27	127	[[C]:1]	[1: 1]	Improvement	Closed
1	28	33	1	24	29	[[C]:1]	[1: 1]	Improvement	Closed

Tab. 42 Wybrane wiersze pliku excel (data.xlsx) cz.2