Politechnika Wrocławska
Wydział Elektroniki
Informatyka

Rozprawa doktorska

Metody zarządzania zapewnianiem jakości oprogramowania wykorzystujace modele predykcji defektów

AUTOR:
Marian Jureczko

Promotor: prof. dr hab. inż. Jan Magott

Wrocław, 2011

Niniejszą pracę dedykuję dwóm osobom,
których obecność przyczyniła się do jej realizacji.

Ukochanej żonie Alinie
oraz
Promotorowi Janowi Magottowi

Spis treści

Spis symboli i skrótów

Skrót	Opis
σ	odchylenie standardowe
r	współczynnik korelacji Pearsona
r_s	współczynnik korelacji rang Spearmana
w	wydanie projektu programistycznego
Mw	model predykcji defektów skonstruowany na podstawie wydania w
M_W	model predykcji defektów skonstruowany na podstawie wydań projektów należących do zbioru W
$E(M, w)$	ocena modelu predykcji defektów M dokonana na wydaniu w
t	statystyka testowa testu–t dla prób zależnych
r_{pb}	siła efektu
W	statystyka testowa testu Shapiro-Wilka
p	prawdopodobieństwo testowe
df	liczba stopni swobody
$F(1, df)$	statystyka F, stosowana w teście Levenea
Λ	statystyka lambda Wilksa

Rozdział 1

Jakość w inżynierii oprogramowania

1.1 Wprowadzenie

W inżynierii oprogramowania używa się kilku różnych definicji jakości. Według Kana [57] najpopularniejsze z nich to zaproponowana przez Crosbiego [15] "zgodność z wymaganiami", zaproponowana przez Gryna [30] "przydatność do użytku" oraz zasugerowany przez samego Kana [57] "brak defektów w produkcie".

Stosowanie definicji "zgodności z wymaganiami" wymaga precyzyjnego zdefiniowania tychże wymagań. Następnie, w trakcie wytwarzania oprogramowania, wykonuje się pomiary zgodności z wymaganiami, a wszelakie odstępstwa od wymagań interpretuje się jako obniżenie jakości produktu końcowego.

Definicja "przydatności do użytku" uwzględnia również oczekiwania klienta. Z uwagi na różnorodność potencjalnych oczekiwań klienta wyróżnia się kilka cech jakości związanych z "przydatnością do użycia". Według [57] dwa najważniejsze z nich to "jakość wykonania" i "jakość zgodności" (z oczekiwaniami). "Jakość wykonania" jest czynnikiem ustalającym wymagania i specyfikację produktu. "Jakość zgodności" natomiast określa w jakim stopniu końcowy produkt odpowiada przyjętej wcześniej specyfikacji.

"Brak defektów w produkcie" to definicja, która jest równocześnie najprostszą miarą zgodności z wymaganiami, gdyż jeżeli oprogramowanie zawiera zbyt wiele defektów (w szczególności odstępstwo implementacji od specyfikacji wymagań należy traktować jako defekt) niemożliwe staje się korzystanie z jego funkcjonalności. Zgodnie z tą definicją jakość mierzyć należy zliczając liczbę lub częstość defektów (liczba defektów przypadająca na jednostkę ilości kodu).

Warto również wspomnieć o normie ISO/IEC 9126-1:2001, która definiuje model jakości oprogramowania. Na model ten składają się następujące charakterystyki [43]:

- funkcjonalność - dopasowanie do jawnie lub niejawnie wyrażonych potrzeb klienta;
- niezawodność - zdolność do poprawnej pracy przez wymagany czas w określonych warunkach;
- użyteczność - zrozumiałość i łatwość użytkowania;
- efektywność - wydajność działania i stopień zużywania zasobów;
- konserwowalność - koszt wykonywania modyfikacji w systemie;
- przenośność - zdolność systemu do zaadoptowania w innym środowisku.

Najbardziej wyczerpującą i wszechstronną definicję jakości oprogramowania zdecydowanie zawiera norma ISO/IEC 9126-1:2001. Jednak z uwagi na tematykę rozprawy, czyli stosowanie modeli predykcji defektów w zapewnianiu jakości oprogramowania, zdecydowanie najwygodniejsza będzie definicja zaproponowana przez Kana [57], mianowicie "brak defektów w produkcie".

1.2 Przegląd działań zorientowanych na jakość oprogramowania

Inżynieria oprogramowania kładzie niebagatelny nacisk na działania mające zapewnić wysoką jakość produktu, jakim jest system informatyczny [27], [33], [57]. Działania te ujawniają się w wielu obszarach procesu wytwarzania oprogramowania.

1.2.1 Testowanie oprogramowania

Testowanie jest najobszerniejszym działaniem związanym z zapewnianiem wysokiej jakości systemu informatycznego pod względem ilości angażowanych zasobów. Zdarza się, że koszty testowania przekraczają 50% całkowitych kosztów wytworzenia systemu informatycznego [80]. Klasyczne procedury i metody testowania zostały już precyzyjnie opisane ([80], [98]) i sformalizowane. Istnieje system certyfikowania organizacji *International Software Testing Qualifications Board* (ISTQB) weryfikujący umiejętność posługiwania się najlepszymi praktykami związanymi z testowaniem oprogramowania jak i norma (IEEE 829 [40]) definiująca zestaw dokumentów, który powinien powstać podczas prawidłowo przeprowadzonej fazy testów. Istnieje wsparcie w postaci programów komputerowych ułatwiające wypełnianie zaleceń wymienionych standardów. W szczególności warto tu wymienić program *AuTester* [68], który umożliwia specyfikowanie i dokumentowanie testów w sposób zgodny z normą IEEE 829 oraz wykonywanie testów w sposób częściowo lub całkowicie automatyczny.

Wraz z pojawieniem się zwinnych metodyk wytwarzania oprogramowania, a w szczególności programowania przez testy [5], zaproponowano nowe podejście do testowania oprogramowania. W podejściu tym odwrócono kolejność aktywności w procesie wytwarzania oprogramowania. W programowaniu przez testy zaczynać należy od przygotowania testów, następnie wykonuje się implementację, a dopiero na końcu należy się zastanowić nad architekturą (modelem) systemu, którą tworzy się w ramach refaktoryzacji. Przykładowa metodyka wytwarzania oprogramowania zgodna z tym paradygmatem została sformalizowana w pracy stanowiącej początek badań związanych z niniejszą rozprawą [50]. Wynikające ze stosowania metodyk zwinnych problemy i korzyści dla jakości oprogramowania zostały wyczerpująco opisane w [94], a przykład próby wdrożenia tych praktyk do dużego projektu informatycznego został przeanalizowany w jednej z prac przygotowanych w ramach opracowywania podstaw do niniejszej rozprawy [44].

W metodykach zwinnych bardzo duży nacisk kładzie się na stronę narzędziową procesu testowania. Programowanie przez testy wymaga aby testy zostały zautomatyzowane. Spełnienie tego wymogu bez posiadania wyspecjalizowanych narzędzi nie jest możliwe. Przegląd dostępnych rozwiązań wspierających automatyzację testów opracowano w [51] oraz [52].

1.2.2 Inspekcje oprogramowania

Inspekcja oprogramowania, często nazywana również przeglądem formalnym, to statyczna analiza kodu programu lub dokumentacji projektowej, która ma na celu wykrycie, identyfikację i usunięcie defektów oraz formalne potwierdzenie jakości produktu. Inspekcja jest zazwyczaj realizowana w postaci trwającego maksymalnie kilka godzin spotkania, na którym autor dokumentu lub fragmentu programu relacjonuje wykonane przez siebie modyfikacje. Reszta osób biorących udział w spotkaniu

ma natomiast za zadanie zrozumieć i zweryfikować poprawność relacjonowanych modyfikacji [33]. Najpopularniejszą formułą inspekcji są, nazwane od nazwiska pomysłodawcy, Inspekcje Fagana.

1.2.3 Modele przyrostu niezawodności

Modele przyrostu niezawodności to modele, które na podstawie historii odnajdywania defektów w procesie testowania szacują liczbę defektów, które jeszcze nie zostały odnalezione. Zazwyczaj modele te opierają się na rozkładzie wykładniczym [57]. Do najpopularniejszych modeli przyrostu niezawodności należą: Model Jelinskiego-Morandy, Model Littlewooda, Model niedoskonałego debugowania Goela-Okumoto, Niejednorodny model procesu Poissona Goela-Okumoto oraz Logarytmiczny model Poissona czasu wykonania Musy-Okumoto [57].

Rozszerzeniem tych modeli jest bazujący na rozkładzie Weibulla model Rayleigha. Model Rayleigha, w odróżnieniu od klasycznych modeli przyrostu niezawodności, wykorzystuje informacje nie tylko z fazy testów, ale z całego procesu wytwarzania oprogramowania. Dzięki temu można szacować liczbę defektów ukrytych w systemie na wcześniejszych etapach prac oraz uzyskać precyzyjniejsze oszacowanie w fazie testów.

Model Rayleigha został opracowany pod kątem zastosowania w kaskadowym procesie wytwarzania oprogramowania. Aktualnie proces ten jest wypierany przez procesy zwinne i iteracyjne, takie jak *eXtreme Programming* czy też *Scrum*. Problem dopasowania Modelu Rayleigha do iteracyjnego procesu wytwarzania oprogramowania nie został jeszcze rozwiązany. Jedną z prób jego rozwiązania jest badanie [45] wykonane w ramach przygotowywania podstaw do niniejszej rozprawy. W badaniu tym analizowano możliwość zastosowania Model Rayleigha do oszacowania liczby defektów ukrytych w projektach wytwarzanych w środowisku akademickim w procesie o krótkich iteracjach, czyli diametralnie różnym od kaskadowego, a zbliżonym do zwinnego. Okazało się, że do rozkładu defektów w czasie można stosunkowo łatwo dopasować model, jednak nie udało się zidentyfikować zależności pomiędzy wyjściem modelu a rzeczywistą liczbą defektów ukrytych. Rezultat taki wskazuje na potrzebę dalszych badań. Natomiast w świetle wyników uzyskanych w [46] i [48], gdzie wykazano, że projekty wytwarzane w środowisku akademickim istotnie się różnią zarówno od projektów komercyjnych jak i projektów otwartych (open-source), zdrowy rozsądek podpowiada, że nie wskazane jest przeprowadzanie dalszych badań na projektach akademickich. Uzyskanie danych do tego typu badań z projektów komercyjnych lub otwartych jest natomiast niezwykle kłopotliwe.

1.2.4 Bezpieczeństwo systemów informatycznych

Istnieje grupa systemów informatycznych, w których bezpieczeństwo odgrywa krytyczną rolę. Są to systemy, których awaria lub nieprawidłowe działanie może doprowadzić do czyjejś śmierci, uszczerbku na zdrowiu lub poważnych strat finansowych. Wyróżniamy dwa rodzaje analiz bezpieczeństwa: probabilistyczne oraz deterministyczne. Stosowane są następujące metody analiz [96]:

- diagramy przyczynowo-skutkowe,
- analiza rodzajów i skutków uszkodzeń,
- analiza drzew zdarzeń,
- analiza hazardu i gotowości systemu,
- analiza drzew niezdatności.

Metoda analizy drzew niezdatności z zależnościami czasowymi, będących rozszerzeniem klasycznych drzew niezdatności, została zaproponowana w [32] i [34], a następnie uszczegółowiona do postaci

algorytmu w [96]. W oparciu o ten algorytm opracowano narzędzie umożliwiające analizę drzew niezdatności z zależnościami czasowymi [53], [54].

1.2.5 Metryki oprogramowania

Definicja 1.2.1 *Metryki oprogramowania to miary pewnej właściwości oprogramowania lub jego specyfikacji. Norma IEEE 1061-1998 [39] definiuje metrykę jako "funkcję odwzorowującą jednostkę oprogramowania w wartość liczbową. Ta wyliczona wartość jest interpretowana jako stopień spełnienia pewnej własności jakości jednostki oprogramowania."*

Metryki oprogramowania pełnią kluczową rolę w niniejszej rozprawie. W związku z tym ich rola została dokładniej omówiona w Rozdziale 1.3.

1.3 Zastosowania metryk oprogramowania

Definicja metryk oprogramowania została już podana w Rozdziale 1.2.5. Warto jednak jeszcze zdefiniować elementy, które przy wyliczaniu i stosowaniu metryk odgrywają kluczową rolę.

Definicja 1.3.1 *Pomiar zgodnie z [33] to proces, w którym atrybutom elementów świata rzeczywistego przydzielane są liczby lub symbole w taki sposób, aby charakteryzować te atrybuty według jasno określonych zasad. Wartości przydzielane atrybutom w ramach tego procesu nazywane są miarą danego atrybutu.*

Definicja 1.3.2 *Proces [33] to każde określone działanie, zbiór działań lub okres czasu w ramach projektu wytwarzania albo eksploatacji oprogramowania. Przykłady obejmują: przygotowanie nowego wydania systemu, projektowanie systemu czy też okresy czasu, takie jak "końcowe sześć miesięcy trwania projektu".*

Definicja 1.3.3 *Produkt [33] to każdy artefakt powstały w wyniku procesu. Przykłady obejmują: specyfikację projektową, model systemu, kod źródłowy, klasę, metodę.*

W działaniach projakościowych stosuje się bardzo różne metryki oprogramowania. Ze względu na rodzaj atrybutów jakie przy pomocy metryki są mierzone możemy podzielić metryki na dwa typy [33], [38]:

- Metryki wewnętrznych atrybutów. Pomiar dokonywany jest wyłącznie w odniesieniu do mierzonego obiektu. Na przykład długość jest wewnętrznym atrybutem pliku z kodem źródłowym programu, który można wyrazić przy pomocy liczby linii kodu. W dalszej części rozprawy, a w szczególności przy konstruowaniu modeli predykcji defektów, wykorzystywane będą metryki tego właśnie typu.

- Metryki zewnętrznych atrybutów. Są to takie metryki, które mogą zostać zmierzone tylko przy uwzględnieniu jak mierzony obiekt odnosi się do innych elementów swojego środowiska. Przykładami takich metryk są niezawodność oraz wydajność.

Z uwagi na cel dokonywania pomiaru można metryki podzielić na trzy klasy [33]:

- Metryki złożoności. Złożoność oprogramowania jest pojęciem wysokiego poziomu, do opisania go potrzeba wielu atrybutów. W związku z tym nie będzie nigdy jednej miary złożoności oprogramowania [21]. Metryk należących do tej klasy zaproponowano już setki. Najszerzej znane z nich to: nauka o programach Halsteada [36] i złożoność cyklomatyczna McCabea [75] jeżeli

chodzi o programy proceduralne oraz zestawy CK [13] oraz QMOOD [3] jeżeli chodzi o programy zorientowane obiektowo. Metryki te są bardzo często wykorzystywane w modelach predykcji defektów, co zostało szerzej omówione w Rozdziale 1.4.

- Metryki szacowania nakładów. Są to metryki reprezentujące atrybuty związane z rozmiarem kodu programu, a wyliczane w celu oszacowania nakładów niezbędnych do zrealizowania systemu informatycznego. Na ich podstawie konstruuje się modele szacowania nakładów. Przykładem takiego modelu jest model COCOMO (ang. Constructive Cost Model).

- Metryki funkcjonalności. Przykładem takiej metryki są punkty funkcyjne Albrechta. Są to metryki odzwierciedlające punkt widzenia użytkownika na zagadnienie funkcjonalności systemu. Wylicza się je zazwyczaj na podstawie specyfikacji systemu.

Henderson-Sellers [38] zaproponował podział metryk z uwagi na typ artefaktu jaki opisują:

- Metryki produktu. Są to metryki, które odzwierciedlają atrybut produktu dla stanu w jakim ten produkt znajdował się w danej chwili czasu. W związku z tym są to metryki, które nic nie mówią o zmienności atrybutów w czasie. Przykładami takich metryk są liczba linii kodu oraz złożoność cyklomatyczna McCabea.

- Metryki procesu. Ten rodzaj metryk obejmuje metryki uwzględniające zmienność atrybutu w czasie. Przykładem takiej metryki jest liczba defektów odnalezionych w zadanym okresie czasu czy też liczba modyfikacji wykonanych przez programistów w zadanym okresie czasu.

Z punktu widzenia niniejszej rozprawy najbardziej adekwatny jest podział zaproponowany przez Hendersona-Selersa. W opisywanych w kolejnych rozdziałach eksperymentach badane będą modele predykcji defektów. Modele te będą wykorzystywały metryki produktu jako stałą bazę niezależną od typu przeprowadzanego badania, a w niektórych eksperymentach będą rozszerzane o metryki procesu w celu zweryfikowania użyteczności tych właśnie metryk w predykcji defektów.

Metryki oprogramowania są często wykorzystywane w inżynierii oprogramowania do szacowania wielkości których bezpośrednio zmierzyć się nie da. Przy pomocy metryk oprogramowania mierzy się zestaw atrybutów systemu informatycznego, a uzyskane wartości wykorzystuje się zazwyczaj do konstrukcji modelu. Model z kolei służy do oszacowania rozważanej, niemierzalnej wielkości. Tym sposobem próbuje się szacować koszty i zasoby potrzebne do zrealizowania systemu (dopóki system nie zostanie zrealizowany wielkości te są niemierzalne) czy też jakość wytworzonego systemu informatycznego. Szczególnym przypadkiem szacowania jakości systemu informatycznego są opisane w kolejnym rozdziale modele predykcji defektów.

1.4 Modele predykcji defektów

Definicja 1.4.1 *Model predykcji defektów to narzędzie często zrealizowane w postaci formuły matematycznej, które na podstawie wartości metryk danego produktu dokonuje oszacowania liczby defektów znajdujących się w tym produkcie.*

Aby poprawnie zinterpretować oszacowanie dostarczane przez model predykcji defektów trzeba dysponować definicją defektu. Norma 982.2 IEEE/ANSI [41] definiuje defekt jako anomalię w produkcie, która może być:

- brakami lub niedoskonałościami znalezionymi we wczesnych etapach prac nad rozważanym systemem informatycznym,

- błędem w funkcjonowaniu produktu, który jest już na tyle dojrzały, że jego działanie można weryfikować przy pomocy testów.

W ramach niniejszej pracy analizowano wiele, w istotny sposób różniących się od siebie projektów. W szczególności różnić się one mogły przyjmowaną podczas prac nad nimi definicją defektu. Aby umożliwić eksperymenty analizujące równocześnie kilka projektów niezbędna była taka definicja defektu, którą można zastosować w każdym z analizowanych projektów. Koru i Liu [66] w oparciu o analizę przeprowadzoną przez Nikorę i Munsona [86] zaproponowali aby defekt definiować jako modyfikację, którą trzeba przeprowadzić w odpowiedzi na problem lub zbiór problemów występujących w oprogramowania. W niniejszej rozprawie posłużono się tą właśnie definicją. Uszczegółowiono ją jednak w sposób uwzględniający uwarunkowania technologiczne:

Definicja 1.4.2 *Defekt to zagadnienie, które zostało zaraportowane i oznaczone jako błąd w systemie śledzenia błędów używanym w projekcie, w którym zagadnienie zostało zidentyfikowane.*

Istnieją różne typy modeli predykcji defektów. Catal i Diri [12] podzielili je ze względu na rodzaj artefaktu, którego dotyczy predykcja na modele poziomu: metody, klasy, komponentu, pliku i procesu. Zaproponowano również podział uwzględniający metodę konstrukcji modelu [12]: modele budowane metodami statystycznymi, modele budowane metodami uczenia maszynowego, modele mieszane wykorzystujące zarówno metody statystyczne jak i nauczanie maszynowe oraz modele mieszane korzystające z metod statystycznych oraz z opinii ekspertów. Wśród metod statystycznych bardzo popularne są techniki oparte o regresję (np.: [11], [77], [89], [115]). Natomiast wśród metod uczenia maszynowego najczęściej wykorzystuje się techniki eksploracji danych takie jak *Random Forrest* (np.: [31], [71]) oraz sieci neuronowe (np.: [62], [61], [104]). Wyczerpujący przegląd prac związanych z predykcją defektów można znaleźć w [12], [64] oraz [90].

Zasady, którymi należy się kierować podczas prac nad modelami predykcji defektów bez względu na sposób konstrukcji modelu, zostały zebrane i opisane przez Fentona w [22]. Fenton przypomina w swojej pracy, że badania dotyczące metryk oprogramowania należą do dyscypliny miernictwa i w związku z tym powinny podlegać ogólnie przyjętym w miernictwie praktykom. Uaktualnienie tych zasad, skoncentrowane na aktualnych trendach w eksperymentach dotyczących predykcji, zostało zaprezentowane przez Gaya i innych w [28]. W pracy tej autorzy koncentrują się na aspektach krytycznych dla możliwości zreprodukowania badania empirycznego, jakim jest konstrukcja i walidacja modelu predykcji defektów. W szczególności podkreślają istotność udostępniania danych, które były podstawą badań i nie ukrywania zastosowanej w eksperymencie procedury. Przedstawione w kolejnych rozdziałach eksperymenty zostały zaprojektowane w duchu tych właśnie zasad.

Najważniejsze prace z zakresu predykcji defektów to między innymi: opracowanie Brianda i innych [9] dotyczące metryk zależności w systemach obiektowych, analiza Basiliego i innych [4] dotycząca użyteczności zestawu metryk Chidambera i Kemerera w predykcji defektów, opracowanie Fentona i Neila [24] omawiające najczęściej popełniane w tej dziedzinie błędy metodyczne czy też eksperyment Brianda i innych [10] analizujący użyteczność różnych metryk produktu w predykcji defektów.

Niniejsza praca koncentruje się na modelach dokonujących predykcji na poziomie klasy i wykorzystujących w procesie budowy modelu metody statystyczne, mianowicie postępującą regresję liniową.

Modele predykcji defektów umożliwiają zarówno oszacowanie jakości systemu informatycznego przez estymowanie liczby znajdujących się w nim defektów jak i wskazanie elementów tego systemu, które są obarczone największą liczbą defektów. Dysponując informacją o rozmieszczeniu defektów można efektywniej wykorzystać budżet przeznaczony na testowanie aplikacji zwiększając zaangażowanie w obszarach systemu wskazanych przez model jako te, które zawierają największą liczbę defektów oraz zmniejszając zaangażowanie w obszarach, które model oznaczył jako wolne od błędów. Jak

podają Boehm i Papaccio [7], w badaniach empirycznych obwiązuje, mająca również zastosowanie w predykcji defektów [18], zasada 80:20. Mała część kodu źródłowego programu (zazwyczaj przybliżana do 20%) jest odpowiedzialna za większość defektów (zazwyczaj przybliżana do 80%). W pracach [113], [114] i [115] pokazano, że można w ten sposób zidentyfikować ponad 80% defektów wskazując zaledwie 20% plików wchodzących w skład badanego systemu. Co oznacza, że korzystając z takiego modelu, można ograniczyć testy do zaledwie 20% plików i mimo tego znaleźć aż 80% defektów ukrytych w testowanym systemie informatycznym.

1.5 Cel i teza pracy

Celem pracy jest zidentyfikowanie czynników, w głównej mierze metryk procesu, które zwiększają dokładność predykcji modelu predykcji defektów zbudowanego na podstawie metryk produktu oraz usprawnienie międzyprojektowej predykcji defektów (ang. corss-project defect prediction). Cel ten ma zostać osiągnięty przez przeprowadzenie szeregu eksperymentów empirycznych i zweryfikowania ich wyników metodami statystycznymi. Identyfikowanie czynników będzie przeprowadzane przez dodawanie ich do modelu zbudowanego na podstawie metryk produktu i weryfikowanie, czy uzyskany w ten sposób model daje lepsze wyniki niż model ich nie wykorzystujący. Usprawnianie predykcji międzyprojektowej będzie wykonane przez zidentyfikowanie takich klastrów projektów, że w obrębie danego klastra będzie można dokonywać predykcji posługując się tym samym modelem. Przez predykcję międzyprojektową należy rozumieć predykcję, w której używa się modelu zbudowanego na podstawie innego projektu (lub zestawu projektów) niż ten, na rzecz którego predykcję się wykonuje.

Tezę pracy można sformułować zatem następująco:

1. **Istnieją czynniki, których dołączenie do modelu predykcji defektów opierającego się na metrykach produktu, spowoduje, że model ten będzie dostarczał dokładniejszych predykcji.**

2. **Istnieją takie klastry projektów informatycznych, że w obrębie klastra będzie można z powodzeniem stosować jeden model predykcji defektów do wszystkich projektów.**

Zidentyfikowanie czynników pozytywnie wpływających na dokładność predykcji pozwoli na konstruowanie modeli obarczonych mniejszym błędem. Dzięki temu będzie można lepiej organizować proces testowania oprogramowania przez koncentrowanie zasobów w tych obszarach testowanego systemu, które rzeczywiście obarczone są największą liczbą defektów. Zidentyfikowanie klastrów projektów pozwoli z kolei na stosowanie predykcji międzyprojektowej, która jest jedyną opcją w projektach, dla których nie gromadzono danych historycznych niezbędnych do skonstruowania modelu predykcji defektów.

1.6 Opis struktury pracy

Pierwszy rozdział pracy traktuje o roli jakości w inżynierii oprogramowania. Zawarto tam przegląd najważniejszych praktyk stosowanych w celu poprawienia jakości systemów informatycznych. Następnie skoncentrowano się na będących w centrum zainteresowań niniejszej pracy metrykach oprogramowania oraz modelach predykcji defektów. Po zarysowaniu tła sformułowano tezę pracy. Pierwszy rozdział zawiera również definicje, fundamentalnych z punktu widzenia badań opisanych w kolejnych rozdziałach, pojęć.

Rozdział 2 zawiera definicje wszystkich wykorzystywanych w badaniach metryk oprogramowania oraz stan wiedzy dotyczący zagadnień analizowanych w niniejszej pracy. Opisano tam najważniejsze badania dotyczące każdego z analizowanych w kolejnych rozdziałach czynników oraz prace badające możliwość stosowania predykcji międzyprojektowej w modelach predykcji defektów.

Rozdział 3 koncentruje się na zbieraniu danych potrzebnych do eksperymentów. Najpierw opisano tam projekty, które były przedmiotem badań. Następnie przedstawiono statystyki opisowe i korelacje z liczbą defektów poszczególnych metryk. Uszczegółowieniem tych statystyk, jest Dodatek A oraz Dodatek B. W rozdziale tym przedstawiono również narzędzia, które w związku z potrzebą zbierania danych do eksperymentów zostały stworzone lub rozbudowane.

W Rozdziale 4 przedstawiono strukturę i wyniki eksperymentu badającego czynniki wpływające na dokładność modeli predykcji defektów. Omówiono tam również zagrożenia dla walidacji uzyskanych wyników i wyciągnięto płynące z tych wyników wnioski.

Eksperymenty dotyczące predykcji międzyprojektowej zostały przedstawione w Rozdziale 5. Przedstawiono tam strukturę eksperymentów uraz uzyskane wyniki. Podobnie jak w poprzednim rozdziale omówiono również zagrożenia dla walidacji uzyskanych wyników i wyciągnięto płynące z tych wyników wnioski.

Rozdział 6 przedstawia przykład wdrożenia modelu predykcji defektów w projekcie przemysłowym. Wdrażany model był w dużej mierze oparty na wynikach eksperymentów opisywanych we wcześniejszych rozdziałach. Rozdział ten stanowi zatem zweryfikowanie badań zaprezentowanych w niniejszej pracy w warunkach przemysłowych.

Podsumowanie uzyskanych wyników zawiera Rozdział 7. Omówiono tam zarówno wkład w stan wiedzy dotyczącej predykcji defektów jaki mają przeprowadzone eksperymenty jak i znaczenie opracowanych na potrzeby badań narzędzi.

Rozdział 2

Przegląd stanu wiedzy o modelach predykcji defektów

W rozdziale tym opisano i zdefiniowano najpopularniejsze metryki oprogramowania, a następnie omówiono najważniejsze prace z zakresu modeli predykcji defektów ze szczególnym uwzględnieniem zagadnień powiązanych z eksperymentami opisanymi w Rozdziałach 4 oraz 5.

2.1 Metryki produktu

Istnieje wiele różnych metryk oprogramowania. Tu omówiono jedynie najważniejsze z nich zaczynając od metryk zaprojektowanych z myślą o programach proceduralnych, a kończąc na metrykach obiektowych. Spośród wymienionych tu metryk w dalszych badaniach zastosowano wszystkie za wyjątkiem miar złożoności Halsteada.

2.1.1 Miary złożoności Halsteada

Halstead [36] założył, że każdy system informatyczny może zostać zrealizowany przez wybranie i zestawienie pewnej liczby jednostek składniowych (leksemów) rozpoznawanych przez kompilator. Jednostki składniowe można następnie podzielić na operatory (funkcje) i operandy (argumenty). Na tej podstawie Halstead zaproponował następujące miary:

- n_1 – liczba różnych operatorów w programie,
- n_2 – liczba różnych argumentów w programie,
- N_1 – liczba wystąpień operatorów,
- N_2 – liczba wystąpień argumentów.

W oparciu o powyższe miary Halstead zdefiniował metryki. Najważniejsze z nich to:

- Zasób słownictwa(n) $n = n_1 + n_2$,
- Długość(N) $N = N_1 + N_2 = n_1 log_2(n_1) + n_2 log_2(n_2)$,
- Wielkość(V) $V = N log_2(n) = N log_2(n_1 + n_2)$,

- Poziom(L) $L = \frac{V^*}{V} = (\frac{2}{n_1}) * (\frac{n_2}{N_2}c)$,
- Trudność(D) $D = \frac{V}{V^*} = (\frac{n_1}{2}) * (\frac{N_2}{n_2})$,
- Nakład pracy(E) $E = \frac{V}{L}$,
- Defekty(B) $B = V/S^*$,

gdzie V^* oznacza najmniejszą ilość kodu reprezentowaną przez jedną wbudowaną funkcję wykonującą zadanie całego programu, zaś S^* to średnia liczba decyzji orzekających o wprowadzeniu lub niewprowadzeniu błędu, według Halsteada wynosi ona 3000.

Prace Halsteada miały ogromny wpływ na późniejsze badania związane z metrykami oprogramowania. Pomimo tego, zaproponowane przez niego metryki są obiektem nieustającej krytyki. Kan podaje [57], że badania empiryczne zwykle nie potwierdzają założeń i definicji Halsteada, za wyjątkiem oszacowań długości programu, której użyteczność jest niezwykle ograniczona, bowiem do jej wyliczenia niezbędna jest znajomość czynników N_1 i N_2. Wartości tych czynników można natomiast poznać dopiero po ukończeniu prac nad programem, a wtedy znana jest również rzeczywista długość programu, więc nie ma powodów aby ją szacować. Z punktu widzenia modeli predykcji defektów komentarza wymaga jeszcze wzór na liczbę defektów. Nie znalazł on niestety potwierdzenia w badaniach empirycznych. Wielkość S^* jest stała, w związku z tym wzór Halsteada oznacza, że liczba defektów jest liniową funkcją wielkości programu. Ignoruje on tym samym zjawisko zróżnicowania gęstości defektów pomiędzy różnymi systemami informatycznymi czy też modułami jednego systemu.

2.1.2 Metryki Chidambera i Kemerera

Chidamber i Kemerer [13] zaproponowali najpopularniejszy aktualnie zestaw metryk obiektowych mierzących złożoność klas. Powszechnie znane są one jako zestaw metryk CK.

Weighted Methods per Class (WMC). Metryka reprezentuje ważoną liczbę metod w klasie. W celu uproszczenia obliczeń nadaje się zazwyczaj wszystkim metodom tą samą wagę (tak też zrobiono w będących przedmiotem tej pracy badaniach). Metryka ta w takiej sytuacji jest więc równa liczbie metod klasy.

Depth of Inheritance Tree (DIT). Metryka ta jest równa liczbie klas–przodków na najdłuższej ścieżce dziedziczenia prowadzącej do rozważanej klasy.

Number Of Children (NOC). Metryka ta jest równa liczbie bezpośrednich potomków (podklas) rozważanej klasy.

Coupling Between Object classes (CBO). Metryka reprezentuje liczbę klas powiązanych z rozważaną klasą. Istotne są zarówno powiązania wchodzące jak i wychodzące. Powiązanie może mieć miejsce przez wywołanie metody, dostęp do atrybutu klasy, dziedziczenie, argument metody, typ zwracany przez metodę oraz wyjątek.

Response For a Class (RFC). Metryka ta jest równa liczbie różnych metod, które mogę być wywołane, gdy instancja obiektu rozważanej klasy otrzymuje komunikat. W odpowiedzi na komunikat mogą zostać wywołane zarówno metody pochodzące z rozważanej klasy, jak i pochodzące z innych klas.

Lack of Cohesion in Methods (LCOM). Metryka ta mierzy kohezję (spójność) klasy. Kohezja wskazuje jak blisko metody danej klasy są powiązane z atrybutami tej klasy. W celu wyliczenia wartości tej metryki należy wygenerować wszystkie możliwe pary metod rozważanej klasy, a następnie od liczby par metod, które współużytkują przynajmniej jeden atrybut klasy odjąć liczbę par, które nie współużytkują żadnego atrybutu klasy.

2.1.3 Zestaw metryk QMOOD

Zestaw obiektowych metryk oprogramowania powszechnie nazywany QMOOD został zaproponowany przez Bansiya i Davisa w [3]. Zestaw ten obejmuje metryki opisujące wiele różny artefaktów wytwarzanych podczas programowania. Tutaj skoncentrowano się tylko na metrykach poziomu klasy.

Number of Public Methods (NPM). Metryka ta zlicza metody, które zostały w rozważanej klasie zadeklarowane jako publiczne. Metryka ta jest również znana pod nazwą *Class Interface Size (CIS)*.

Data Access Metric (DAM). Metryka ta reprezentuje stosunek liczby atrybutów prywatnych (ang. private) i zabezpieczonych (ang. protected) do liczby wszystkich atrybutów występujących w rozważanej klasie.

Measure Of Aggregation (MOA). Metryka ta, jak wskazuje nazwa, jest miarą agregacji. Jej wartość jest równa liczbie atrybutów, które są instancjami klas, które zostały zdefiniowane przez programistę tworzącego rozważany system informatyczny.

Measure of Functional Abstraction (MFA). Metryka jest równa stosunkowi liczby metod odziedziczonych przez rozważaną klasę do liczby wszystkich metod tej klasy. Konstruktory i metody odziedziczone po *java.lang.Object* są ignorowane.

Cohesion Among Methods of class (CAM). Metryka obrazuje pokrewieństwo pomiędzy metodami rozważanej klasy. Równa jest sumie różnych typów występujących jako parametry poszczególnych metod rozważanej klasy podzielonej przez iloczyn różnych typów parametrów wszystkich metod rozważanej klasy i liczby metod tej klasy.

2.1.4 Zorientowane na jakość rozszerzenie zestawu CK

Zestaw metryk CK został przeanalizowany przez Tanga, Kao oraz Chena [103]. Wynikiem tej analizy były cztery metryki, które zestaw ten mają za zadanie uzupełniać. Opisano jedynie trzy z tych metryk. Czwarta ma zastosowanie tylko do tych języków programowania, w których nie występuje mechanizm odśmiecania (ang. garbage collector). W niniejszej pracy badane są tylko programy napisane w języku Java, który w mechanizm odśmiecania jest wyposażony. W związku z tym czwarta metryka nie ma tu zastosowania.

Inheritance Coupling (IC). Metryka ta reprezentuje liczbę klas-przodków powiązanych z rozważaną klasą. Klasa jest powiązana ze swoim przodkiem jeżeli przynajmniej jedna z odziedziczonych metod funkcjonalnie zależy od nowej lub przedefiniowanej metody. Dokładniej rzecz ujmując klasa jest powiązana ze swoim przodkiem gdy przynajmniej jeden z poniższych warunków jest spełniony:

- jedna z odziedziczonych metod używa atrybutu, który jest zdefiniowany w nowej lub przedefiniowanej metodzie,

- jedna z odziedziczonych metod woła przedefiniowaną metodę,
- jedna z odziedziczonych metod jest wołana przez przedefiniowaną metodę i używa parametru, który jest zdefiniowany w tej przedefiniowanej metodzie.

Coupling Between Methods (CBM). Wartość tej metryki jest równa liczbie nowych i przedefiniowanych metod, które są powiązane z odziedziczonymi metodami. O powiązaniu można mówić, gdy chociaż jeden z warunków wymienionych w metryce IC jest spełniony.

Average Method Complexity (AMC). W celu wyliczenia wartości tej metryki należy obliczyć średni rozmiar metody w rozważanej klasie. Rozmiar metody jest równy liczbie linii kodu binarnego (ang. Java binary code) w tej metodzie.

2.1.5 Metryki zależności Martina

Robert Martin zaproponował w [73] dwie bardzo popularne metryki obrazujące zależności między klasami obiektowego systemu informatycznego.

Afferent coupling (Ca). Wartość tej metryki jest równa liczbie klas, które od rozważanej klasy zależą. Zależność jest tu rozumiana tak samo jak powiązanie w przypadku metryki CBO. Jednak CBO, w odróżnieniu od Ca, zlicza zarówno zależności wychodzące jak i wchodzące do rozważanej klasy. Metryka Ca uwzględnia tylko zależności wchodzące.

Efferent couplings (Ce). Wartość tej metryki jest równa liczbie klas, od których rozważana klasa zależy. Zależność jest tu rozumiana tak samo jak powiązanie w przypadku metryki CBO. Jednak CBO, w odróżnieniu od Ce, zlicza zarówno zależności wychodzące jak i wchodzące do rozważanej klasy. Metryka Ce różni się od metryki Ca kierunkiem zależności jaki jest w niej zliczany, gdyż uwzględnia ona tylko zależności wychodzące.

2.1.6 Pozostałe metryki produktu

Obok wymienionych powyżej metryk w modelach predykcji defektów stosuje się jeszcze często trzy inne metryki.

Lack of Cohesion in Methods (LCOM3). Metryka ta jest unormowaną wersją metryki zaproponowanej przez Chidamber i Kemerer [13] i wylicza się ją ze wzoru 2.1. We wzorze tym m reprezentuje liczbę metod w klasie, a liczbę atrybutów w klasie, $\mu(A)$ liczbę metod, które używają atrybutu A. Metryka ta została zaproponowana przez Hendersona-Sellersa [38].

$$LCOM3 = \frac{(\frac{1}{a}\sum_{j=1}^{a}\mu(A_j)) - m}{1 - m} \tag{2.1}$$

McCabe's Cyclomatic Complexity (CC). W celu wyliczenia wartości tej metryki należy skonstruować graf ilustrujący wszystkie ścieżki przepływu sterowania w rozważanej metodzie. Wartość metryki jest wtedy równa liczbie różnych ścieżek w grafie zwiększonej o jeden. Metryka ta jest metryką metody, natomiast w badanych w niniejszej pracy modelach predykcji defektów stosowane są metryki klasy. W związku z tym zdefiniowano dwie metryki klasy bazujące na metryce CC:

- **Max(CC)** – największa wartość metryki CC pośród wszystkich metod należących do rozważanej klasy,
- **Avg(CC)** – średnia arytmetyczna metryki CC wyliczona dla wszystkich metod wchodzących w skład rozważanej klasy.

Metryka została zaproponowana przez McCabe [75].

Lines of Code (LOC). Metryka ta jest równa liczbie linii kodu badanej klasy. W celu wyliczenia wartości tej metryki brane są pod uwagę linie kodu binarnego skompilowanych klas Javy. Często w badaniach związanych z modelami predykcji defektów stosuje się liczbę linii kodu źródłowego, zamiast kodu binarnego. Jednak jak podają Fenton i Neil [24] te dwie wersje metryki LOC są silnie ze sobą skorelowane i w związku z tym należy je traktować jako alternatywę. Zawsze wybierać tylko jedną z nich. Z punktu widzenia modeli predykcji defektów obie są jednakowo wartościowe [24].

2.2 Metryki procesu

W rozdziale tym zgromadzono definicje wszystkich metryk procesu, które były analizowane w opisanych w Rozdziale 4 eksperymentach. Prace, które badają możliwość stosowania tych metryk w modelach predykcji defektów są omówione w Rozdziale 2.3.

Number of Revisions (NR). Wartość tej metryki jest równa liczbie modyfikacji (rewizji) danej klasy wykonanych w ramach opracowywania danego wydania projektu. Metryka ta odzwierciedla zatem stabilność kodu danej klasy. Klasy, które nie są stabilne cechują się dużą liczbą modyfikacji i przez to wysoką wartością metryki NR.

Number of Distinct Commiters (NDC). Metryka ta reprezentuje liczbę różnych programistów, którzy modyfikowali mierzoną klasę w ramach opracowywania danego wydania projektu. Metryka ta może zatem przyjmować niskie wartości nawet dla często modyfikowanych klas, ale pod warunkiem, że za każdym razem modyfikuje je ten sam programista. Przy pomocy tej metryki można odzwierciedlić typ własności kodu. Jeżeli klasa ma jednego właściciela to wartość metryki NDC pozostanie niemal zawsze niska. Natomiast w przypadku współdzielonych klas wartość tej metryki będzie zazwyczaj wysoka.

Number of Modified Lines (NML). W celu wyliczenia tej metryki należy przeanalizować historię zmian w rozważanej klasie. Wartość metryki jest równa sumie wszystkich dodanych oraz usuniętych w ramach opracowywania danej wersji projektu linii kodu. Metryka ta była wyliczana na podstawie danych historycznych z systemu kontroli wersji (CVS). Należy więc tu zaznaczyć, że system kontroli wersji nie operuje pojęciem modyfikacji linii kodu. Dla niego modyfikacja to usunięcie starej wersji linii i dodanie nowej wersji. W taki też sposób modyfikacja była interpretowana podczas wyliczania tej metryki.

Is New (IN). Jest to metryka dwuwartościowa. Przyjmuje ona wartości *new* (reprezentowana czasami przy pomocy liczby 1) oraz *old* (reprezentowana czasami liczbą 0). Wartość *new* oznacza, że rozważana klasa jest nowa, czyli powstała w ramach opracowywania ostatniego wydania projektu. Wartość *old* oznacza, że klasa jest stara, czyli powstała w ramach opracowywania jednego z wcześniejszych wydań projektu.

Number of Defects in Previous Version (NDPV). Metryka ta jest równa liczbie defektów, które zostały wykryte i naprawione w rozważanej klasie w ramach prac nad danym wydaniem systemu. Są to więc defekty pochodzące z wcześniejszego okresu czasu, niż ten którego dotyczy predykcja wykonana przy pomocy modelu predykcji defektów.

Number of Evening Revisions (NER). Jest to metryka, która podobnie jak NR bierze pod uwagę liczbę modyfikacji wykonanych w rozważanej klasie w ramach opracowywania danego wydania projektu. Tutaj brane są jednak pod uwagę tylko te modyfikacje, które były wykonywane pod presją kończącego się dnia roboczego. Z uwagi na geograficzne rozproszenie zespołów realizujących projekty otwarte nie było możliwe wskazanie pory dnia, którą dla tych zespołów można kojarzyć ze zbliżającym się końcem pracy. W związku z tym metrykę tą wyliczano tylko dla projektów przemysłowych.

Number of Pre-code-freeze Revisions (NPR). Metryka ta została dopasowana do procesu wytwarzania oprogramowania stosowanego w projektach przemysłowych badanych w Rozdziałach 4 i 5. W projektach tych cykl związany z przygotowaniem nowego wydania systemu trwa sześć miesięcy i jest podzielony na dwie fazy: programowania i testowania. Programowanie kończy się zamrożeniem kodu (ang. code-freeze), po którym rozpoczyna się testowanie. Podczas testowania nie implementuje się już nowych funkcjonalności. Jedyne modyfikacje kodu, jakie wtedy mogą być wykonywane, to naprawy krytycznych defektów odkrytych podczas testów. Zastosowanie takiego procesu wytwarzania oprogramowania powoduje, że największa presja związana ze zbliżającymi się terminami występuje bezpośrednio przed zamrożeniem kodu. Wartość metryki NPR jest zatem równa liczbie modyfikacji wykonanych w rozważanej klasie w trakcie trzech tygodni bezpośrednio poprzedzających zamrożenie kodu do danego wydania badanego projektu.

2.3 Czynniki wpływające na predykcję defektów

Badane w niniejszej rozprawie czynniki to w głównej mierze metryki procesu. Analizowano tylko jeden czynnik, który nie był metryką procesu, mianowicie rozmiar. W celu zwiększenia przejrzystości tego rozdziału podzielono go na części odpowiadające poszczególnym czynnikom. W każdej części zostały omówione prace dotyczące tylko jednego z nich.

Metryki procesu nie zostały jeszcze tak dokładnie przebadane jak metryki produktu. W związku z tym część z nich nie doczekała się jeszcze jednoznacznych interpretacji. Różni badacze przyjmują nieznacznie różniące się od siebie definicje. W rozdziale tym opisano nie tylko prace posługujące się identycznymi z przedstawionymi w Rozdziale 2.2 definicjami metryk. Uwzględniono również takie badania, gdzie autorzy do poszczególnych metryk podchodzili w nieco inny sposób.

Metryki procesu, stanowiące zdecydowaną większość badanych w niniejszej rozprawie czynników mogących poprawić predykcję, stają się coraz powszechniej stosowanym przez badaczy narzędziem. Fenton [25] wpierw dostarczył konstruktywnej krytyki modeli opierających się metryki produktu, a następnie zaproponował [23] model oparty tylko na metrykach procesu i metrykach projektu. Z drugiej jednak strony pojawiają się również prace [77], [78] podważające wyniki uzyskane przez Fentona i wykazujące dużą użyteczność metryk produktu.

Number of Revisions (NR). Informacja o liczbie historycznych wersji kodu źródłowego była już badana w wielu pracach. Jednymi z najciekawszych pod względem jakości uzyskanych predykcji są opracowania przygotowane przez Bella, Ostranda i Weyuker [6], [60], [87], [88], [113], [114], [115], [116]. W pracach tych wykorzystywano w modelach predykcji defektów pierwiastek kwadratowy z liczby wersji kodu źródłowego powstałych w trakcie prac nad poprzednią wersją projektu. Do konstrukcji

modelu stosowano regresję binominalną. Uzyskane modele cechowały się bardzo dużą dokładnością predykcji. Najlepsze z nich pozwalały na zidentyfikowanie do 92% defektów w zaledwie 20% modułów. Niebanalną rolę w tych modelach odgrywała metryka *NR*. Dodatkowo wart podkreślenia jest fakt, że w pracach tych stosowano taką samą definicję tej metryki jak w niniejszej rozprawie. Analizując uzyskane modele autorzy doszli do wniosku, że wzrost liczby historycznych wersji zwiększa wartość wyjścia modelu, czyli estymowaną liczbę defektów. W [88] stwierdzili wręcz, że moduły modyfikowane w poprzednim wydaniu miały około trzy razy więcej defektów niż te, które zmianom nie podlegały.

Do podobnych wniosków doszli Graves, Karr, Marron i Siy [29] analizując mający wieloletnią historię, przemysłowy projekt programistyczny. Projekt ten pochodził z branży telekomunikacyjnej, składał się z 1,5 miliona linii kodu i był modyfikowany przez kilkuset programistów. W oparciu o takie dane autorzy stwierdzili, że historia projektu zawiera o wiele bardziej przydatne informacje niż te, które można uzyskać z metryk opisujących rozmiar i strukturę, czyli metryk produktu. W szczególności odkryli, że stosowanie liczby linii kodu (najpopularniejsza w modelach predykcji defektów metryka rozmiaru) nie ma żadnego sensu jeżeli do modelu wprowadzi się informację o liczbie wersji (metryka *NR*). Odkrycie to bazowało na fakcie, że wprowadzenie liczby linii kodu do modelu, który już zawierał metrykę *NR* nie poprawiało uzyskiwanych predykcji nawet w najmniejszym stopniu. Z uwagi na fakt, że autorzy przebadali tylko jeden projekt, trudno ocenić możliwość generalizowania uzyskanych wyników.

Ciekawe badania wykorzystujące informacje o liczbie historycznych wersji kodu przeprowadzili Schröter, Zimmermann, Premraj i Zeller. W [93] przebadali oni systemy kontroli wersji i śledzenia defektów projektu *Eclipse*. Wyliczyli korelacje metryk procesu z liczbą awarii zaraportowanych przed datą wydania oraz po dacie wydania nowej wersji systemu. W przypadku awarii zgłoszonych przed wydaniem korelacja z metryką reprezentującą liczbę historycznych modyfikacji była największa dla wszystkich badanych przypadków. Współczynnik korelacji Pearsona był równy $0,37$ i $0,47$ (analizowano dwie wersje projektu), a współczynnik korelacji rang Spearmana $0,44$ oraz $0,56$. Korelacje z liczbą awarii zgłoszonych po wydaniu były jednak już zdecydowanie niższe. W przypadku korelacji Pearsona były to wartości $0,14$ i $0,19$, a w przypadku korelacji rang Spearmana $0,15$ oraz $0,17$.

Prawdopodobnie najistotniejsze z punktu widzenia liczby przebadanych projektów badania nad metryką *NR* zostały przeprowadzone przez Illes-Seifert i Peach [42]. W pracy tej przebadano aż dziewięć różnych programów, wszystkie należały do klasy projektów otwartych. W oparciu o te dziewięć projektów badano różne metryki procesu, między innymi *częstotliwość zmian* (ang. frequency of change). Metryka ta w oczywisty sposób jest powiązana z badaną w niniejszej rozprawie metryką *NR*. Autorzy odkryli istnienie wysokiej korelacji pomiędzy *częstotliwością zmian*, a liczbą defektów. Wynosiła ona od $0,431$ do $0,641$. Stosowano korelację rang Spearmana. Na podstawie tych obserwacji zarekomendowano metrykę *częstotliwość zmian* jako bardzo dobry predykator, który z powodzeniem może zostać zastosowany do wskazania klas obarczonych największą liczbą defektów.

Shihab i in. [95] przebadali projekt *Eclipse* w celu wskazania jak najmniej licznego zbioru metryk, które mogłyby posłużyć do konstrukcji skutecznego modelu predykcji defektów. Łącznie analizowano 34 metryki produktu i procesu (w tym odpowiednik metryki *NR*). W celu wyłonienia zestawu optymalnych metryk iteracyjnie stosowano krokową regresję logistyczną. Ostatecznie otrzymano model składający się z zaledwie czterech metryk, pozostałe 30 odrzucono. Wśród czterech wyselekcjonowanych metryk znalazła się również metryka będąca odpowiednikiem *NR*. Uzyskany w ten sposób model dawał predykcje niegorsze niż model skonstruowany z wykorzystaniem wszystkich metryk.

Moser, Pedrycz i Succi [79] przeprowadzili eksperyment, w którym porównywali użyteczność metryk procesu i metryk produktu. Wśród analizowanych metryk procesu znalazła się między innymi metryka o nazwie *REVISIONS* zdefiniowana w taki sam sposób jak badana w niniejszej pracy metryka *NR*. Do budowy modeli stosowano trzy różne metody: drzewa decyzyjne, regresję logistyczną oraz

naiwne klasyfikatory bayesowskie. Eksperyment przeprowadzono na trzech wersjach projektu *Eclipse*. Aby porównać użyteczność poszczególnych metryk konstruowano osobno modele dla metryk procesu, dla metryk produktu oraz modele hybrydowe wykorzystujące wszystkie metryki. Dobre wyniki uzyskano zarówno dla modeli wykorzystujących metryki procesu jak i dla modeli hybrydowych. Zdecydowanie gorszych predykcji dostarczały modele wykorzystujące tylko metryki produktu. Dodatkowo warto zauważyć, że autorzy wskazali metrykę *REVISIONS* jako jedną z dwóch najwartościowszych spośród metryk procesu.

Ciekawe rozszerzenie metryki *NR* zaproponowali Nagappan i in. w [85]. W pracy tej zdefiniowano metrykę reprezentującą całe serie zmian. Następnie korzystając z regresji logistycznej skonstruowano model predykcji defektów, który zastosowano w projektach *Windows Vista* oraz *Eclipse*. Okazało się, że model wykorzystujący tą nową metrykę dawał najlepsze predykcje spośród wszystkich badanych modeli. Pozostałe badane modele nie wykorzystywały metryk reprezentujących serie zmian. Autorzy tej pracy, w jednej ze swoich wcześniejszych prac [82], wykorzystywali również klasyczną wersję metryki *NR*.

Knab, Pinzger i Bernstein w [65] przeprowadzili eksperyment dotyczący konstruowania modeli predykcji defektów z wykorzystaniem drzew decyzyjnych dla siedmiu wersji projektu *Mozilla*. Oprócz metryk produktu używano dwóch metryk blisko związanych z metryką *NR* (*Number of modification reports* i *Number of shared modification reports*) oraz kilku innych metryk procesu. Przeprowadzono eksperymenty, w których za każdym razem używano innych metryk do konstruowania drzewa decyzyjnego. W eksperymencie, w którym uzysakno najlepsze wyniki użyto wszystkich metryk za wyjątkiem tych związanych z historycznymi defektami. W uzyskanym drzewie najważniejszą rolę odgrywały metryki związane z metryką *NR*, aczkolwiek drzewo to dostarczało poprawnych predykcji jedynie w 62% przypadków. Mimo tego autorzy uznali, że byli w stanie dokonywać predykcji defektów w oparciu o liczbę historycznych modyfikacji.

Khoshgoftaar i inni [60] przeanalizowali duży system informatyczny z branży telekomunikacyjnej napisany w języku Pascal. Wyliczono statystyki występowania błędów w modułach zmodyfikowanych oraz niezmodyfikowanych. Okazało się, że w zmodyfikowanych występuje średnio $3,22$ błędów w module, a w niezmodyfikowanych zaledwie $0,22$.

Metryka *NR* jest z powodzeniem stosowana w licznych eksperymentach. Wykazano, że jest ona skorelowana z liczbą defektów oraz, że można stosując model wykorzystujący tą metrykę z powodzeniem wykonywać skuteczną predykcję występowania defektów. Niewiele jest natomiast prac porównujących modele wykorzystujące tą metrykę z modelami, w których ta metryka nie występuje. Porównanie takie wykonano w pracach [29] oraz [79], ale zostało ono przeprowadzone na podstawie tylko jednego projektu, więc nadużyciem byłoby generalizowanie przedstawionych tam wyników. Modele porównywane były również przez Bella, Ostranda i Weyuker [6], [87], [88], [113], [114], [115]. Ale w pracach tych nie dość, że również badano niereprezentatywny zbiór projektów (ich liczba rosła jedynie do pięciu i wszystkie one były wytwarzane przez tą samą firmę w podobnym procesie) to jeszcze na dodatek porównywane modele na tyle się od siebie różniły, że trudno było wyciągać wnioski dotyczące jedynie jednej metryki.

Number of Distinct Commiters (NDC). Metryka reprezentująca liczbę różnych programistów, którzy modyfikowali dany element systemu była intensywnie wykorzystywana w modelach konstruowanych przez Bella, Ostranda i Weyuker [6], [87], [88], [113], [114], [115], [116]. Najciekawsze z tych prac to [114], [115] oraz [116]. W [114] i [115] została przeanalizowana istotność metryki *NDC*. Dodanie do modelu informacji o liczbie różnych programistów poprawiło predykcję, ale jedynie w nieznacznym stopniu. Bez metryki *NDC* modele były w stanie wskazać od $76,4\%$ do $93,8\%$ defektów w 20% plików. Natomiast z tą metryką modele wskazywały od $76,4\%$ do $94,8\%$ defektów. Poprawa predykcji

kształtowała się zatem na poziomie ułamka procentu. Badania te przeprowadzono na trzech dużych projektach przemysłowych, które posiadały wieloletnią historię rozwoju. Rozwinięcie tych prac w kierunku uszczegółowienia zależności pomiędzy liczbą defektów a aktywnościami poszczególnych programistów zostało przedstawione w [116]. W pracy tej próbowano znaleźć zależność pomiędzy częstością występowania defektów a konkretnym programistą. Niestety zależności takiej nie wykryto i to pomimo faktu, iż wielu programistom zaangażowanym w prace nad projektami programistycznymi jej istnienie wydaje się być oczywiste.

Zależność pomiędzy liczbą defektów a aktywnościami poszczególnych programistów była również badana przez Matsumoto i innych w [74]. W odróżnieniu od wcześniej wspomnianej pracy, Matsumoto się takiej zależności doszukał. Można jednak kwestionować zaproponowaną przez niego metodę identyfikowania programisty odpowiedzialnego za wprowadzenie defektu do systemu. Założył on mianowicie, że odpowiedzialnym za wprowadzenie defektu jest zawsze ten programista, który był autorem kodu, który został zmodyfikowany w ramach naprawiania defektu.

Liczba różnych programistów była również wykorzystywana przez Zimmermanna, Nagappan, Galla, Gigera i Murphiego w [119]. W pracy tej budowano modele predykcji defektów dla dwunastu projektów otwartych i przemysłowych. Do konstrukcji modeli używano regresji logistycznej. Niestety cele tej pracy nie zawierały ewaluacji metryk i w związku z tym nie analizowano ani tego na ile dobrym predykatorem defektów jest metryka *NDC*, ani tego czy dodanie jej do modelu poprawia dokładność predykcji.

Korelacja pomiędzy metryką *NDC* a awaryjnością systemu została wyliczona i przeanalizowana przez Schrötera, Zimmermanna, Premraja i Zellera w [93]. W badaniach tych opierano się na projekcie *Eclipse*. Badano zarówno korelację z liczbą awarii zaraportowanych przed datą wydania jak i liczbą awarii zaraportowanych po dacie wydania nowej wersji systemu. W zależności od typu awarii korelacja ta wynosiła $0,15$ i $0,41$ w przypadku korelacji Pearsona oraz 0.13 i $0,49$ w przypadku korelacji rang Spearmana. Na podstawie tych wyników można wyciągnąć wniosek, że metryka *NDC* jest dobrym predykatorem awarii zgłaszanych przed oficjalnym wydaniem. Trudno natomiast zawyrokować czy metryka ta nadaje się do estymowania awarii zgłoszonych po wydaniu. Warto w tym miejscu przypomnieć, że w analizach będących głównym tematem niniejszej rozprawy bada się liczbę defektów, a nie awarii (aczkolwiek istnieje ścisła zależność pomiędzy istnieniem defektówa a występowaniem awarii – awarie są wywoływane przez defekty) oraz nie wprowadza się rozróżnienia na defekty zgłoszone przed i po wydaniu.

Podobne wyniki jak Schröter uzyskali Illes-Seifer i Paech w [42]. W pracy tej przebadano dwanaście projektów otwartych i między innymi wyliczono korelację rang Spearmana pomiędzy liczbą różnych autorów a liczbą defektów. Dla większości przebadanych projektów była ona stosunkowo wysoka, od $0,352$ do $0,741$. Wyjątkiem był projekt *Jmol* dla którego wspomniana wyżej korelacja wyniosła zaledwie $0,16$.

Jedynym empirycznym badaniem, które zakwestionowało użyteczność metryki *NDC* w predykcji defektów było to przeprowadzone przez Gravesa, Karra, Marrona oraz Siy [29]. Analizowano tam duży system z branży telekomunikacyjnej. Do predykcji defektów stosowano uogólniony model liniowy. Uzyskane wyniki pozwoliły autorom na stwierdzenie, że brak dowodów empirycznych potwierdzających tezę, że duża liczba programistów pracujących nad modułem systemu informatycznego powoduje lub jest powiązana z wysoką awaryjnością tego modułu.

Odpowiednik metryki *NDC* o nazwie *AUTHORS* był wykorzystywany w [79] przez Mosera, Pedrycza oraz Succiego. W pracy tej porównywano skuteczność metryk procesu i metryk produktu w predykcji defektów na podstawie trzech wersji projektu *Eclipse*. Uzyskane wyniki pozwoliły na stwierdzenie istnienia przewagi metryk procesu nad metrykami produktu. Jednak metryka *AUTHORS* nie została wskazana jako jedna z najwartościowszych spośród przeanalizowanych.

Nagappan, Murphy i Basili w [84] badali możliwość wykorzystania metryk opisujących strukturę organizacji w predykcji defektów. Pośród innych analizowali również metykę reprezentującą liczbę programistów (ang. Number of Engineers). Do konstrukcji modelu używano regresji logistycznej. Dokładność predykcji została oceniona przy pomocy dwóch metryk: *precyzja* (ang. *precision*) i *pamięć* (ang. *recall*). *Precyzja* została zdefiniowana jako odsetek poprawnie zakwalifikowanych klas (model dokonywał predykcji binarnej: klasa zawiera defekty albo klasa jest wolna od defektów) spośród klas wskazanych przez model jako zawierające błędy *Pamięć* to natomiast stosunek poprawnie zakwalifikowanych klas zawierających defekt do wszystkich klas, które zawierały defekt. Zarówno *precyzja* jak i *pamięć* były najwyższe w przypadku modelu opartego o metryki opisujące organizację i wynosiły odpowiednio: $86,2\%$ oraz $84,0\%$. Badano system *Windows Vista*.

Ratzinger, Pinzger oraz Gall przeanalizowali w [92] łącznie 63 różne metryki (w tym *NDC*) w pięciu przemysłowych projektach programistycznych. Do konstrukcji modeli predykcji defektów używano regresji liniowej, drzew decyzyjnych i klasyfikatorów. Wyjście uzyskanych modeli było skorelowane z rzeczywistą liczbą defektów na poziomie od $0,490$ do $0,614$, a metryka *NDC* odgrywała w tych modelach istotną rolę – autorzy opublikowali fragmenty uzyskanych modeli.

Metryka *NDC* jak i metryki do niej bardzo zbliżone były jak do tej pory intensywnie badane i wykorzystywane w modelach predykcji defektów. Zauważono, że jest ona skorelowana z liczbą defektów [42], [93] oraz stosowano ją w dokonujących wysokiej jakości predykcji modelach [6], [74],[84], [87], [88], [92], [113], [114], [115], [116], [119]. Istnieje jednak również praca kwestionująca, w oparciu o badania empiryczne, użyteczność tej metryki [29]. Przeprowadzono prace porównujące skuteczność modeli zawierających tą metrykę z modelami jej pozbawionych [79]. Trudno jednak uznać wyniki tam przedstawione za rozstrzygające.

Number of Modified Lines (NML). Analiza rozmiaru modyfikacji jako predykatora defektów była głównym tematem badania przeprowadzonego przez Laymana, Kudrajavetsa oraz Nagappana w [69]. Autorzy zdefiniowali cztery metryki zależące od liczby zmodyfikowanych linii. *Produkt* (ang. *churn*) reprezentujący sumę nowych bloków (ang. new blocks) oraz zmodyfikowanych bloków (ang. changed blocks). *Produkt względny* (ang. *relative churn*) będący stosunkiem *produktu* do liczby wszystkich bloków (ang. total blocks). *Produkt usunięty* (ang. *deleted churn*) będący stosunkiem liczby usuniętych bloków (ang. deleted blocks) do liczby wszystkich bloków. *Produkt NCD* (ang. *NCD churn*) będący stosunkiem *produktu* do liczby usuniętych bloków. Następnie wykorzystano te cztery metryki łącznie z zestawem metryk produktu i utworzono nowe zmienne wykorzystując analizę głównych składowych (PCA). Korzystając z tych nowych zmiennych przy pomocy postępującej regresji logistycznej skonstruowano model predykcji defektów. Tak skonstruowany model okazał się być dobrym predykatorem defektów, jego *dokładność* wahała się od $73,3\%$ do $86,7\%$. *Dokładność* to procent poprawnie zaklasyfikowanych artefaktów. W badaniach analizowano jeden, średniego rozmiaru projekt stworzony przez MicroSoft.

Purushothaman i Perry [91] analizowali liczbę zmodyfikowanych linii kodu w napisanym w C/C++ i składającym się z 50 podsystemów programie z branży telekomunikacyjnej. Opisali rozkład rozmiaru modyfikacji dla poszczególnych plików. Opis ten wskazał, że połowa zmian w systemie była efektem modyfikacji nie przekraczających 10 linii kodu. Następnie modyfikacje zostały zestawione z występowaniem defektów. Okazało się, że modyfikacje nie przekraczające 10 linii kodu były powiązane z prawdopodobieństwem wprowadzenia defektu równym 8%. Wraz ze wzrostem rozmiaru modyfikacji rosło również prawdopodobieństwo wprowadzenia defektu. Dla modyfikacji dotyczących od 20 do 50 linii prawdopodobieństwo to wynosiło 19%, a dla modyfikacji przekraczających 50 linii wynosiło aż 35%.

Graves, Karr, Marron i Siy [29] na podstawie analizy dotyczącej jednego dużego systemu informatycznego stwierdzili, że liczba historycznych modyfikacji jest zdecydowanie lepszym predykatorem defektów niż szeroko stosowana metryka *LOC*. Co więcej, uznali, że metryka ta jest wprost proporcjonalna do liczby defektów. Następnie zbudowali model predykcji defektów, w którym obok liczby historycznych modyfikacji użyli metryki reprezentującej wiek kodu źródłowego. Uzyskany w ten sposób model okazał się być jednym z najlepszych jakie autorom udało się skonstruować.

Metryka ta była również analizowana i wykorzystywana (jako jedna z wielu) przez Nagappana i Balla w [81], gdzie stwierdzono, że jest ona bardzo dobrym predykatorem gęstości występowania defektów w dużych projektach przemysłowych. Stwierdzenie to zostało następnie dokładnie zweryfikowane we wspomnianej na początku tego paragrafu pracy [69]. Użyteczność tej metryki potwierdza fakt, że była ona stosowana w dalszych pracach tych autorów. W pracy [82] wykorzystano ją podczas analizy systemu *Windows Server 2003*, a w [84] i [85] podczas analizy systemu *Windows Vista*. Natomiast w pracy [119] wykorzystano ją w eksperymencie dotyczącym aż dwunastu projektów przemysłowych i otwartych.

Dziewięć różnych metryk ściśle powiązanych z metryką *NML* był wykorzystywany w [79] przez Mosera, Pedrycza oraz Succiego. W pracy tej porównywano skuteczność metryk procesu i metryk produktu na podstawie trzech wersji projektu *Eclipse*. Uzyskane wyniki pozwoliły na stwierdzenie istnienia przewagi metryk procesu nad metrykami produktu. Jednak metryki związane z *NML* nie zostały wskazane jako najwartościowsze.

Sześć dużych projektów otwartych przebadał Hassan w [37] w celu zweryfikowania użyteczności metryki związanej z historycznymi modyfikacjami. Na podstawie charakterystyki występowania modyfikacji zdefiniowano entropię. Entropia była tym większa im bardziej równomiernie modyfikacje były rozłożone w systemie. Następnie wykazano, że moduły systemu mające większą entropię były powiązane z większą liczbą defektów. Praca opierała się na tezie mówiącej, że im bardziej kompleksowa modyfikacja tym większa szansa na wystąpienie defektu w modyfikowanym pliku. Do niemal takich samych wniosków doszli Śliwerski, Zimmermann i Zeller [97] badając projekty *Eclipse* oraz *Mozilla*. Mianowicie stwierdzili oni na podstawie historii tych projektów, że im większa modyfikacja tym większe prawdopodobieństwo, że wprowadza ona błąd.

Ratzinger, Pinzger oraz Gall przeanalizowali w [92] łącznie 63 różne metryki (w tym *NML*) w pięciu przemysłowych projektach programistycznych. Do konstrukcji modeli predykcji defektów używano regresji liniowej, drzew decyzyjnych i klasyfikatorów. Wyjście uzyskanych modeli było skorelowane z rzeczywistą liczbą defektów na poziomie od $0,490$ do $0,614$, a metryka *NML* odgrywała w tych modelach istotną rolę – autorzy opublikowali fragmenty uzyskanych modeli.

Metryki związane z liczbą zmodyfikowanych linii stają się coraz popularniejsze. Przeprowadzane są badania definiujące coraz bardziej wyrafinowane metryki oparte o historię modyfikacji (w szczególności warto tu wymienić prace [37] oraz [85]). Co szczególnie ciekawe, badania takie prowadzą zazwyczaj do bardzo dobrych wyników. Niewiele jest jednak analiz porównawczych, które pozwoliłyby rozstrzygnąć czy te dobre wyniki są wynikiem zastosowania optymalnych metryk, czy też po prostu analizowane dane miały strukturę, która pozwalała na uzyskanie dobrych wyników bez względu na rodzaj stosowanych metryk.

Is New (IN). Metryka *IN* jest przykładem metryki związanej z wiekiem kodu. Metryki odzwierciedlające wiek klas są bardzo popularne, ale niestety są definiowane w bardzo różnorodny sposób. Bell, Ostrand i Weyuker [87], [88] używali metryki przyjmującej dwie wartości: *nowy plik* oraz *stary plik*, czyli identycznej z używaną w niniejszej rozprawie. Niemal identycznej definicji używali również Khoshgoftaar i in. w [59]. W tej pracy wprowadzono jednak jeszcze drugą metrykę związaną z wiekiem kodu źródłowego. Ta druga metryka mogła przyjmować trzy różne wartości reprezentujące odpowied-

nio nowe moduły, moduły, które były nowe w poprzednim wydaniu systemu oraz pozostałe. Podobnie zdefiniowanej metryki używali Illes-Seifer i Paech [42]. W ich pracy jednak posługiwano się zupełnie innymi nazwami dla poszczególnych wartości przyjmowanych przez tą metrykę, mianowicie: *nowo narodzony*, *młody* i *stary*. Istnieje również kilka różnych definicji metryki reprezentującej wiek kodu. Bell, Ostrand i Weyuker w [6] przez wiek kodu rozumieli liczbę miesięcy, przez które dany plik istniał w systemie, ale już w [113] i [115] wiek kodu definiowali jako liczbę wcześniejszych wydań systemu, w których dany plik występował. Moser, Pedrycz oraz Succi [79] wiek kodu zdefiniowali jako liczbę tygodni, przez którą dany plik istniał w systemie. Dodatkowo autorzy ci używali ważonego modyfikacjami wieku. Podobnie do zagadnienia podeszli Graves, Karr, Marron i Siy, którzy w [29] wiek kodu interpretowali jako średnią z dat modyfikacji ważoną przez rozmiar poszczególnych modyfikacji.

Metryki związane z wiekiem kodu były niemal we wszystkich badaniach uznawane za dobre predykatory defektów. Jednakże naturę relacji pomiędzy wartościami przyjmowanymi przez tą metrykę a liczbą defektów trzeba uznać za nieznaną z uwagi na fakt, że w różnych badaniach dokonywano w tej materii sprzecznych ze sobą odkryć.

Graves, Karr, Marron i Siy [29] zbudowali model predykcji defektów, w którym obok metryki reprezentującej wiek użyli liczby historycznych modyfikacji. Uzyskany w ten sposób model okazał się być jednym z najlepszych jakie autorom udało się skonstruować.

Khoshgoftaar i inni [60] przeanalizowali duży system informatyczny z branży telekomunikacyjnej napisany w języku Pascal. Wyliczono między innymi statystyki występowania błędów w modułach nowych oraz w całym systemie. Okazało się, że w nowych występuje średnio $3,43$ błędów w module, kiedy w skali całego systemu jest to średnio zaledwie $1,93$.

Moser, Pecrycz i Succi w [79] wykorzystywali dwie metryki związane z wiekiem – *AGE* oraz *WEIGHTED_AGE*. W pracy tej porównywano skuteczność metryk procesu i metryk produktu na podstawie trzech wersji projektu *Eclipse*. Uzyskane wyniki pozwoliły na stwierdzenie istnienia przewagi metryk procesu nad metrykami produktu. Jednak metryki związane z wiekiem nie zostały wskazane jako najwartościowsze.

Bell, Ostrand i Weyuker [87], [88] wykorzystali z powodzeniem metrykę związaną z wiekiem kodu w budowanych modelach predykcji defektów i zauważyli, że w każdym z przebadanych przez nich wydań projektów informatycznych, odsetek nowych plików zawierających defekty był wyższy niż odsetek starych plików zawierających defekty. Obserwacja ta została potwierdzona w kolejnych pracach tych autorów: [6], [113], [115]. Trzeba jednak mieć na uwadze fakt, że w pracach tych autorów obiektem badań były zawsze podobne do siebie projekty, co w sporym stopniu ogranicza możliwość generalizowania przedstawionych tam wyników.

Khoshgoftaar i in. w [59] uznali, że istotność stosowanej przez nich metryki dwuwartościowej jest bardzo zmienna. Natomiast w przypadku metryki trójwartościowej zidentyfikowali jej ujemną korelację z liczbą defektów. Ujemna korelacja została zinterpretowana jako czynnik wskazujący na wyższą jakość modułów od dawna istniejących w systemie.

Zupełnie inne wyniki zostały uzyskane przez Illes-Seifer i Paech w [42]. Tam co prawda potwierdzono istnienie korelacji z liczbą defektów, ale jej wartość była zupełnie inna. Stwierdzono mianowicie, że pliki zaliczane do grup *nowo narodzony* oraz *młody* wcale nie są źródłem największej liczby defektów.

Przytoczone badania wskazują, że wyniki stosowania metryk zbliżonych do metryki *IN* nie są jednoznaczne. W zależności od tego jakie projekty badano czasami wskazywano, że największą liczbą defektów są obciążone stare klasy [42], a czasami że nowe [59]. Taki stan rzeczy uzasadnia dalsze badania poświęcone tej metryce.

Number of Defects in Previous Version (NDPV). Informacje o historycznych defektach są jedną z najczęściej wykorzystywanych w predykcji defektów metryk procesu. Przykładem tego typu prac są

wspomniane we wprowadzeniu modele przyrostu niezawodności. Adaptację modelu przyrostu niezawodności do klasycznej predykcji defektów przeprowadzili Wahyudin i in. w [110]. W pracy tej wykorzystywano informacje o defektach z poprzedniego wydania projektu do predykcji w kolejnym wydaniu.

Próby stosowania metryki *NDPV* do predykcji defektów podejmowano już w latach 80-tych minionego stulecia. Yu, Shen i Dunsmore [118] przeanalizowali dwa duże komercyjne projekty i na tej podstawie stwierdzili, że istnieje silna liniowa zależność pomiędzy liczbą defektów odkrytych na wczesnych fazach rozwoju projektu, a tymi które zostały znalezione później. Ponadto doszli do wniosku, że prawdopodobieństwo wykrycia kolejnych defektów jest wprost proporcjonalne do liczby defektów już wcześniej w danej sekcji odkrytych.

Arisholm i Briand [2] rozważali liczbę defektów naprawionych w wydniu $n-1$, naprawionych w wydaniu $n-2$ jak i kilka innych metryk w dużym, obiektowo zorientowanym, cały czas ewoluującym systemie informatycznym. Autorzy zbudowali wielowymiarowy model predykcji defektów stosując regresję logistyczną na poddanym transformacji logarytmicznej wartościom poszczególnych metryk. Uzyskany w taki sposób model pozwolił na uzyskanie poprawnej predykcji w ponad 80% badanych przypadków.

Bell, Ostrand i Weyuker [89], [113] i [115] konstruowali modele predykcji defektów na podstawie kilku projektów przemysłowych. Modele budowano przy pomocy regresji binomialnej, a jedną z wykorzystywanych metryk był pierwiastek kwadratowy z liczby defektów zidentyfikowanych w ramach prac nad poprzednim wydaniem projektu. W taki sposób konstruowane modele okazały się być bardzo dobrymi predykatorami defektów. Przy ich pomocy można było wykryć ponad 80% defektów w zaledwie 20% plików składających się na badany system.

Graves, Karr, Marron i Siy [29] na podstawie analizy dotyczącej jednego dużego systemu informatycznego pochodzącego z branży telekomunikacyjnej skonstruowali kilka modeli dokonujących predykcji liczby awarii. Najprostszy ze skonstruowanych modeli wykorzystywał jedynie jedną metrykę, mianowicie liczbę awarii w poprzednim wydaniu systemu. Według autorów skonstruowanie modelu dającego dokładniejsze predykcje niż ten najprostszy okazało się być nie lada wyzwaniem bez względu na to jakich metryk by nie wykorzystywali.

Kim, Zimmermann, Whitehead i Zeller [63] przeanalizowali siedem otwartych projektów informatycznych i opracowali algorytm dokonujący predykcji liczby defektów w oparciu o mechanizm pamięci podręcznej. Dwie naczelne reguły tego algorytmu są powiązane z metryką *NDPV*: *lokalność czasowa* oraz *lokalność czasowo-przestrzenna*. Za *lokalnością czasową* kryje się założenie, że encje które były źródłem defektów w niedalekiej przyszłości, prawdopodonie będą nimi znowu w przyszłości. U podstaw *lokalności czasowo-przestrzennej* legło natomiast przeświadczenie, że jeżeli jakaś encja była źródłem defektów w niedalekiej przeszłości, to jest bardzo możliwe, że encje z jej otoczenia staną się źródłem defektów w przyszłości. Okazało się, że stosowanie tego algorytmu umożliwiło wykrywanie aż od 73% do 95% defektów w zaledwie 10% plików, wskazanych przez model jako obaczonych największą liczbą defektów.

Khoshgoftaar i in. w [59] przebadali wojskowy system informatyczny czasu rzeczywistego. Na podstawie uzyskanych wyników doszli do wniosku, że istnieje zależność pomiędzy występowaniem defektów w przeszłości, a ich pojawianiem się w przyszłości. Mianowicie stwierdzili, że moduły, które były źródłem defektów, będą prawdopodobnie nadal ich źródłem.

Ostrand i Weyuker w [87] analizowali trwałość defektów pomiędzy kolejnymi wydaniami projektu. Znaleźni pewne dowody empiryczne potwierdzające tą tezę, ale nawet zdaniem samych autorów nie były one do końca przekonujące. Okazało się, że od 17% do 54% plików zaklasyfikowanych jako wysoce awaryjne w wydaniu n było tak samo klasyfikowanych w wydaniu *n+1*.

Shihab i in. [95] przebadali projekt *Eclipse* poszukując metryk, które są najwartościowszymi składnikami modeli predykcji defektów. Łącznie analizowano 34 metryki produktu i procesu w tym metrykę reprezentującą liczbę historycznych defektów. W celu wyłonienia zestawu optymalnych metryk iteracyjnie stosowano krokową regresję logistyczną. Metryka będąca odpowiednikiem metryki *NDPV* była jedną z najwartościowszych. Została odrzucona dopiero w ostatniej iteracji, czyli w sytuacji w której model składał się zaledwie z pięciu metryk.

Gyimothy, Ferenc i Siket wyliczyli korelacje pomiędzy liczbami defektów występujących w poszczególnych wersjach projektu. Przebadali *Mozzilę* w wersjach od 1.0 do 1.6. Uzyskane korelacje były zaskakująco wysokie, wahały się od $0,69$ do aż $0,90$. Wynik taki sugeruje, że metryka *NDPV* powinna bardzo dobrze się sprawdzać w modelach predykcji defektów. W szczególności metryka ta powinna dobrze uzupełniać metryki produktu, które zazwyczaj są słabiej skorelowane z liczbą defektów. Badanie to jednak opierało się tylko na jednym projekcie i niestety jego wynik nie był zgodny z tym co uzyskiwano w innych pracach.

Illes-Seifert i Peach wyliczyli w [42] korelacje pomiędzy liczbami defektów w poszczególnych wydaniach dziewięciu różnych projektów otwartych. Przeprowadzony eksperyment doprowadził autorów do wniosku, że nie ma istotnej korelacji pomiędzy badanymi wielkościami. Podobne rezultaty uzyskali Schröter, Zimmermann, Premraj i Zeller w [93]. Analizowali oni korelację trzech różnych metryk z liczbą awarii udokumentowanych przed i po oficjalnym wydaniu w projekcie *Eclipse*. Okazało się, że metryka reprezentująca liczbę historycznych defektów była najsłabiej skorelowana spośród wszystkich trzech przebadanych metryk.

Moser, Pecrycz i Succi w [79] wykorzystywali metrykę *BUGFIXES*, której definicja jest bardzo podobna do definicji metryki *NDPV*. W pracy tej prównywano skuteczność metryk procesu i metryk produktu na podstawie trzech wersji projektu *Eclipse*. Uzyskane wyniki pozwoliły na stwierdzenie istnienia przewagi metryk procesu nad metrykami produktu. Dodatkowo metryka *BUGFIXES* została zalecona jako metryka, którą należy wykorzystywać w pierwszej kolejności podczas wyszykiwania defektów. Do jeszcze bardziej jednoznacznych wniosków doszli Śliwerski, Zimmermann i Zeller w [97] badając projekty *Eclipse* oraz *Mozilla*. Zauważyli mianowicie, że modyfikacje wynikające z naprawiania znalezionych wcześniej defektów są trzy razy częściej źródłem nowych defektów niż pozostałe modyfikacje.

Ratzinger, Pinzger oraz Gall przeanalizowali w [92] łącznie 63 różne metryki (w tym *NDPV*) w pięciu przemysłowych projektach programistycznych. Do konstrukcji modeli predykcji defektów używano regresji liniowej, drzew decyzyjnych i klasyfikatorów. Wyjście uzyskanych modeli było skorelowane z rzeczywistą liczbą defektów na poziomie od $0,490$ do $0,614$, a metryka *NDPV* odgrywała w tych modelach istotną rolę.

W większości przytoczonych prac uznano metrykę reprezentującą historyczne defekty za skuteczny predykator i wartościowy składnik modeli predykcji defektów. Istnieją jednak również badania dotyczące licznego zbioru projektów, w których żadnych zależności tego typu nie udało się doszukać. Brak jednoznacznych wyników w dotychczasowych badaniach jest motywacją do przeprowadzenia dalszych, wykorzystujących szerszy zestaw projektów.

Number of Evening Revisions (NER). Metryka *NER* została po raz pierwszy przebadana w eksperymencie opisanym w Rozdziale 4. Można się jednak doszukać pracy, w której badana jest metryka do pewnego stopnia podobna do *NER*. Śliwerski, Zimmermann i Zeller w [97] badając projekty *Eclipse* oraz *Mozilla* sprawdzali, czy dzień tygodnia, w którym wykonywano modyfikacje jest w jakiś sposób powiązany z prawdopodobieństwem wystąpienia defektu. Okazało się, że najbardziej prawdopodobne jest wprowadzenie defektów w piątkowych modyfikacjach. Różnica pomiędzy poszczególnymi dniami nie była jednak duża. Średnie prawdopodobieństwo wprowadzenia defektu wyniosło $10,4\%$, a naj-

większe, piątkowe wynosiło 12, 2% w projekcie *Eclipse* oraz odpowiednio 41, 5% i 46, 2% w projekcie *Mozilla*. Niemniej wynik taki wskazuje na występowanie pewnego stresu związanego z kończącym się czasem pracy i będącym jego efektem niedbalstwem. W [97] stres ten analizowano w cyklu tygodniowym, metryka *NER* analizuje go natomiast w cyklu dniowym.

Number of Pre-code-freeze Revisions (NPR). Metryka *NPR* została w niniejszej pracy zdefiniowana w sposób ściśle dopasowujący ją do cyklu produkcyjnego projektu w którym była badana. Z uwagi na wysoki stopień dopasowania metryki do konkretnego projektu nie istnieją badania, które posługiwałyby się identyczną metryką. Warto jednak zauważyć, że Illes-Seifert i Peach w [42] zdefiniowali i przebadali podobną metrykę. W pracy tej czas pomiędzy dwoma wydaniami projektu został podzielony na pięć faz: *hotFix* – początkowe 5% czasu pomiędzy wydaniami; *postRelease* – chronologicznie następna faza obejmująca kolejne 10% czasu pomiędzy wydaniami; *preRelease* – 10% czasu pomiędzy wydaniami bezpośrednio poprzedzające fazę *lastMinuteFix*; *lastMinuteFix* – końcowe 5% czasu poprzedzające wydanie; *moderation* – okres czasu pomiędzy fazą *postRelease*, a *preRelease*. Następnie wyliczono korelacje pomiędzy liczbą defektów a liczbą modyfikacji wykonanych w poszczególnych fazach. Na podstawie uzyskanych wyników autorzy stwierdzili, że liczba defektów rośnie wraz z liczbą modyfikacji wykonanych w fazie *preRelease*. Co jednak bardzo ciekawe, zależność ta nie była już słuszna dla fazy *lastMinuteFix*, a wydawać by się mogło, że to właśnie ta końcowa faza jest związana z największym pośpiechem i to właśnie wtedy programistom mogłoby brakować czasu na dotrzymywanie wysokich standardów pracy. Okres czasu rozważany w metryce *NPR* możnaby w pewnym przybliżeniu uznać za sumę faz *preRelease* oraz *lastMinuteFix*.

Rozmiar. Metryki związane z rozmiarem są jednymi z najpopularniejszych w badaniach związanych z modelami predykcji defektów. Istnieje jednak stosunkowo niewiele badań, w których próbuje się wykonać bardziej zaawansowaną analizę, niż zwykłe dodanie metryki związanej z rozmiarem do modelu i uznanie, że wraz ze wzrostem rozmiaru zwiększa się szansa na wystąpienie błędu. Skrajnym przykładem tego typu eksperymentu jest praca Mende i Koschke [76], gdzie skonstruowano model wykorzystujący tylko jedną metrykę, mianowicie *LOC*. Zbudowany w taki sposób model okazał się być zaskakująco skuteczny. Wyjście modelu było mocno skorelowane z liczbą defektów. Korelacja rang Spearmana wynosiła w poszczególnych przypadkach od 0, 41 do 0, 90. Takie postawienie sprawy nie daje jednak nazbyt ciekawych wyników. Jak słusznie zauważa Kitchenham [64], każda dodatkowa linia programu, to dodatkowa okazja do popełnienia błędu, a w związku z tym stwierdzenie liniowej zależności pomiędzy rozmiarem a liczbą defektów nie jest wielkim odkryciem. Tym bardziej ciekawe wydają się więc być badania przeprowadzone przez Fentona i Neila w [24], gdzie odkryto, że duże moduły często zawierają zdecydowanie mniej defektów niż małe. Z drugiej jednak strony istnieją również badania, gdzie dochodzono do zupełnie przeciwnych wniosków [105].

Koru i Liu [66] zauważyli dwa istotne z punktu widzenia rozmiaru zjawiska. Mianowicie, doszli do wniosku, że dla małych modułów metryki produktu przyjmują zazwyczaj niewielkie wartości, a często są wręcz równe zero. W związku z tym charakterystyka małych modułów wykazuje niewielką wariancję i w efekcie tego, podczas konstrukcji modelu trudno jest wychwycić różnicę pomiędzy małymi modułami wolnymi od defektów, a małymi modułami obarczonymi defektami. Równocześnie autorzy ci zauważyli, że w przypadku dużych modułów wariancja metryk jest zdecydowanie większa, ale za to modułów takich jest zazwyczaj stosunkowo niewiele. Mała liczba modułów również może stwarzać problemy podczas konstrukcji modelu. Koru i Liu [66] analizowali pięć realizowanych w NASA projektów, które zostały opublikowane w publicznie dostępnym repozytorium PROMISE i doszli do wniosku, że wyniki predykcji dla zbiorów zawierających wiele małych modułów są zdecydowanie gorsze niż dla pozostałych. W świetle uzyskanych wyników autorzy zalecają podzielenie zbioru da-

nych, dla których ma zostać wykonana predykcja pod względem rozmiaru modułów. Dzięki takiemu podziałowi zdaniem autorów można skonstruować lepiej dopasowane do danych modele i tym samym uzyskać lepsze predykcje. Autorzy zarekomendowali również zwiększanie rozmiaru modułu będącego obiektem predykcji w przypadku gdy jest on wyjątkowo mały. W opisanym przez nich eksperymencie zwiększono rozmiar modułu przez zmianę z predykcji poziomu metody do predykcji poziomu klasy. Eksperyment ten został następnie rozszerzony w [67], gdzie badano projekt Mozilla. Badanie to potwierdziło rolę rozmiaru w predykcji defektów. Dodatkowo autorzy zauważyli użyteczność monitorowania rozwoju pod kątem zmian rozmiaru poszczególnych modułów systemu i pokazali, że można tym sposobem dokonywać predykcji jeszcze przed ukazaniem się pierwszego wydania projektu.

Wpływ rozmiaru na modele predykcji defektów był również badany przez El Emam, Benlarbiego, Goela i Rai [19]. Badano tam, czy metryką *LOC* można w jakimś stopniu wyjaśnić niedokładności predykcji modelu opartego o metryki produktu. Porównywano modele wykorzystujące wybraną metrykę produktu z modelami wykorzystującymi wybraną metrykę produktu oraz rozmiar wyrażony przy pomocy metryki *LOC*. Modele konstruowano przy pomocy regresji logistycznej. Dodatkowo wyliczono korelacje pomiędzy metrykami produktu, a rozmiarem. Na podstawie uzyskanych wyników, postawiono tezę, że metryki produktu są obciążone przez rozmiar. Co z kolei pozwoliło autorom na stwierdzenie, że rozmiar powinien być zawsze w modelach predykcji defektów wykorzystywany. Dzięki temu będzie on użyty jawnie w modelu, a metryki produktu nie będą przez niego w takim stopniu obciążone. Wyniki te, mimo iż bardzo ciekawe, trudno uznać za rozstrzygające. Autorzy przebadali bowiem tylko jeden projekt informatyczny. Ponadto słuszność tych wyników została wpierw zakwestionowana przez Evanco [20], a później przez Madeyskiego [72]. Autorzy ci wskazali, że w pracy [19] przyjęto nie do końca poprawną definicję czynnika zakłócającego, a celem badania było stwierdzenie, czy rozmiar jest czynnikiem zakłócającym. Wątpliwości budzi również założenie mówiące o wyłącznym stosowaniu analizy jednowymiarowej do walidacji metryk.

Zalecenia dotyczące rozmiaru przedstawione w [19] zastosowali w praktyce Subramanyam i Krishnan [102]. Badali oni duży projekt przemysłowy realizowany w Javie oraz C++. Analiza dotyczyła stosowania metryk Chidambera i Kemerera oraz rozmiaru (rozumianego jako liczba linii kodu) do predykcji defektów. W pracy tej weryfikowano między innymi hipotezę mówiącą o tym, że duże klasy są zazwyczaj powiązane z większą liczbą defektów. Autorzy uznali, że hipoteza ta jest prawdziwa z uwagi na fakt, że potwierdziła ją postać uzyskanych modeli predykcji defektów.

Metryki reprezentujące rozmiar są stosowane w predykcji defektów niemal od początków tej dyscypliny. Stosowanie rozmiaru do przewidywania liczby defektów było postulowane już prze Halsteada [36] w 1977 roku. Z drugiej jednak strony tezę mówiącą o różnej zależności pomiędzy metrykami a liczbą defektów zaczęto stawiać stosunkowo niedawno [24], [66]. W efekcie czego nie udało się jeszcze uzyskać wyników istotnych statystycznie. Warto tu jeszcze nadmienić, że prawie wszystkie wspomniane powyżej prace do mierzenia rozmiaru wykorzystywały metrykę LOC. Metrykę WMC, jako metrykę rozmiaru rozważano natomiast w [10].

2.4 Predykcja międzyprojektowa

Jeszcze do niedawna badania dotyczące predykcji międzyprojektowej były bardzo odległe od udzielenia jednoznacznej odpowiedzi na pytanie o możliwość wykonywania takiej predykcji. Gunes Koru oraz Hongfang Liu [66] analizując ten problem doszli do wniosku, że dla różnych środowisk wytwarzania oprogramowania należy konstruować różne modele predykcji defektów każdorazowo uwzględniając indywidualne wzorce występowania defektów. Równocześnie badacze ci uznali jednak, że powyższe sformułowanie nie może prowadzić do konkluzji jakoby tworzenie uogólnionych modeli predykcji defektów (mających zastosowanie do więcej niż jednego projektu) było niemożliwe. Co więcej wyartykułowali

oni zasadność stosowania uogólnionych modeli wskazując, że są one jedyną opcją dla projektów nie dysponujących danymi historycznymi niezbędnymi do konstrukcji modelu predykcji defektów.

Do innych wniosków doszli Wahyudin, Ramler i Biffl. W pracy [109] zaproponowali oni bibliotekę służącą do predykcji defektów. W kontekście definiowania struktury tej biblioteki analizowano możliwość ponownego wykorzystywania danych historycznych pochodzących z różnych projektów. Rozważania te doprowadziły autorów do wniosku, że model predykcji defektów powinien być konstruowany w kontekście konkretnego projektu. Uznali również, że zazwyczaj predykator uzyskany dla jednego projektu, nie nadaje się do zastosowania w drugim. Trudno jednak uznać przedstawione w tej pracy konkluzje za rozstrzygające, ponieważ istnieją badania empiryczne (najważniejsze z nich omówiono poniżej), których wyniki sugerują coś zupełnie innego.

Interesujące badania bliskie predykcji międzyprojektowej zostały przeprowadzone przez Bella, Ostranda i Weyuker. W pracy [89] opisali oni eksperyment przeprowadzony na dwóch dużych projektach przemysłowych, które składały się odpowiednio z 17 i 9 wydań. Zastosowano regresję binominalną oraz metryki wyliczane z kodu źródłowego jak i historię modyfikacji i występowania błędów do skonstruowania modelu predykcji defektów. Praca ta została następnie rozszerzona w [113], przez przeanalizowanie trzeciego projektu. W projekcie tym zastosowano taki sam model jak w obydwu wcześniejszych i uzyskano bardzo zachęcające wyniki. Okazało się, że predykcje uzyskane z tego modelu pozwalały na wykrycie aż 83% wszystkich znajdujących się w projekcie defektów w zaledwie 20% plików składających się na ten projekt. Dalsze badania zostały przedstawione w [115], gdzie liczba badanych projektów została zwiększona do czterech. Przeprowadzone badania pozwoliły autorom na stwierdzenie, że metoda konstrukcji modeli predykcji defektów, którą zaproponowali z myślą o systemach przemysłowych mających wiele historycznych wydań, sprawdza się dobrze nie tylko w tych systemach, dla których była projektowana, ale również w systemach, które wcześniejszych wydań nie miały. Wniosek taki pokazuje, że przy pewnych ograniczeniach, wyniki uzyskane w jednym projekcie mogą z powodzeniem być stosowane w innych projektach. Należy tu jednak pamiętać, że wszystkie projekty badane przez Bella, Ostranda i Weyuker były wytwarzane przez tą samą firmę w ramach zbliżonych procesów wytwarzania oprogramowania i należały do podobnych domen. Granice przenoszenia doświadczeń między projektami zostały tu więc bardzo zawężone. Dodatkowo należy jeszcze nadmienić, że przenoszona pomiędzy projektami była metoda tworzenia modelu (zestaw wykorzystywanych metryk, procedura uczenia modelu) a nie sam model. Natomiast celem badań opisanych w Rozdziale 5 jest przenoszenie modelu między projektami.

Przenoszenie modelu między projektami było analizowane przez Watanabe, Kaiya i Kaijiri w przeprowadzonym przez nich studium przypadku [111]. Przebadali oni dwa podobne do siebie projekty programistyczne: *JEdit* oraz *Sakura Editor*. Oba mają zbliżony rozmiar i należą do tej samej dziedziny, mianowicie oba są edytorami tekstu. Różniły się te projekty natomiast językiem programowania, w którym były realizowane. *JEdit* został napisany w Javie, a *Sakura Editor* w C++. Dla każdego z projektów z osobna skonstruowano model predykcji defektów. Ostatecznie, w celu zweryfikowania możliwości dokonywania predykcji międzyprojektowej, model skonstruowany dla projektu *JEdit* został zastosowany w projekcie *Sakura Editor*, a model skonstruowany dla projektu *Sakura Editor* został zastosowany w projekcie *JEdit*. Wyniki takiej predykcji międzyprojektowej porównano następnie z wynikami predykcji wewnątrzprojektowej, czyli takiej gdzie ten sam projekt był wpierw wykorzystywany do budowy modelu, a następnie dokonywano w nim predykcji defektów przy użyciu tego właśnie modelu. Dokładność predykcji została oceniona przy pomocy dwóch metryk: *precyzja* (ang. *precision*) i *pamięć* (ang. *recall*). W predykcji wewnątrzprojektowej uzyskano *precyzję* $0,828$ i $0,733$ oraz *pamięć* $0,897$ i $0,792$. Natomiast w predykcji między projektowej *precyzja* była równa $0,872$ i $0,622$ a *pamięć* $0,596$ i $0,402$. Uzyskano więc zdecydowanie lepsze wyniki dla predykcji wewnątrzprojektowej niż dla międzyprojektowej, ale mimo to autorzy wyciągnęli wniosek, że w przypadku zbliżonej domeny

i rozmiaru badanych projektów można stosować predykcję międzyprojektową. Jednocześnie jednak przyznali, że wyników badania które przeprowadzili, czyli dotyczącego zaledwie dwóch projektów, nie można uogólniać. W celu uzyskania rozstrzygających wyników należałoby eksperyment powtórzyć na większej grupie projektów.

Zaawansowane badania dotyczące predykcji międzyprojektowej zostały opisane przez Turhana, Menziesa, Benera i Distefano w [108]. W pracy tej przebadano dziesięć różnych wersji projektów programistycznych. W oparciu o uzyskane wyniki stwierdzono, że nie istnieje jeden zestaw metryk produktu, który mógłby posłużyć do konstruowania modeli predykcji defektów w dowolnym projekcie. W przeprowadzonych eksperymentach używano do oceny skuteczności modelu *prawdopodobieństwa wykrycia defektu* (ang. *probability of detection, pd*) oraz *prawdopodobieństwa fałszywego alarmu* (ang. *probability of false alarm, pf*). Warto tu nadmienić, że model dokonywał predykcji binarnej: moduł zawiera defekty albo moduł jest wolny od defektów. Wartości zarówno *pd* jak i *pf* były zdecydowanie wyższe w przypadku predykcji międzyprojektowej niż wewnątrzprojektowej. Wynik taki nie pozwala na rozstrzygnięcie, czy predykcja międzyprojektowa dała wyniki chociażby zbliżone do predykcji wewnątrzprojektowej. Z drugiej jednak strony autorzy wykryli w podzbiorach danych ciekawe zależności, dzięki którym udało im się zaproponować filtrowanie oparte na sąsiedztwie. Sąsiedztwo było definiowane na podstawie euklidesowej odległości pomiędzy wartościami poszczególnych metryk. Zastosowanie filtrowania doprowadziło do istotnego zmniejszenia wartości *pf* w predykcji międzyprojektowej. Niestety autorzy nie zdecydowali się na przeprowadzenie dodatkowej analizy, która pozwoliłaby na zidentyfikowanie metryk lub cech projektu, które decydują o sukcesie predykcji międzyprojektowej. Natomiast pozytywny wpływ filtrowania opartego o sąsiedztwo świadczy o tym, że takie cechy istnieją.

Eksperyment ten został następnie rozszerzony w [107] przez dodanie do zestawu badanych projektów dwóch kolejnych, składających się z trzech wydań, co dało w efekcie całkowitą liczbę 13 wydań. Nowe projekty, w odróżnieniu od wcześniej analizowanych, należały do klasy projektów otwartych. Przeprowadzone analizy potwierdziły wcześniejsze obserwacje dotyczące predykcji międzyprojektowej. Mianowicie w części przypadków okazała się ona być w podobnym stopniu skuteczna co predykcja wewnątrzprojektowa. W szczególności dotyczy to projektów otwartych. Wykorzystanie danych z projektów przemysłowych do konstrukcji modelu, który następnie posłużył do przeprowadzenia predykcji w projektach otwartych dało wyniki równie dobre jak w przypadku predykcji wewnątrzprojektowej.

Prawdopodobnie najbardziej zaawansowany eksperyment z zakresu predykcji międzyprojektowej został przeprowadzony w MicroSoft Research i opisany w [83]. W pracy tej sprawdzano, czy predykatory wyciągnięte z historii jednego projektu mają zastosowanie w drugim. Autorzy przeanalizowali pięć przemysłowych projektów. Analiza wykazała, że nie ma jednego zestawu metryk, który mógłby zostać zastosowany w każdym z pięciu badanych projektów. Aczkolwiek modele predykcji defektów uzyskane z podobnych projektów funkcjonowały dobrze między tymi projektami. Nie podano niestety precyzyjnej definicji podobieństwa projektów. Autorzy analizowali problem predykcji międzyprojektowej przez budowę modelu w oparciu o dane z jednego, wybranego projektu, a następnie używali tego modelu w czerech pozostałych projektach. Następnie porównywano korelacje pomiędzy estymowaną liczbą defektów uzyskaną z modelu, a rzeczywistą liczbą defektów. Analiza ta wykazała, że model skonstruowany w oparciu o historię jednego projektu bardzo rzadko może zostać z powodzeniem zastosowany do wykonania predykcji w innym projekcie.

Eksperyment został w istotny sposób rozszerzony w [119], gdzie opisano przeprowadzenie łącznej liczby 622 predykcji międzyprojektowych dotyczących 12 projektów przemysłowych i otwartych (ang. open-source). Predykcji dokonywano pomiędzy każdymi dwoma wydaniami badanych projektów przy zachowaniu ograniczenia zakazującego dokonywania predykcji w wydaniu starszym niż to, na podstawie którego budowano model. W eksperymencie oceniano skuteczność predykcji przy pomocy *precyzji* (ang. *precision*), *pamięci* (ang. *recall*) i *dokładności* (ang. *accuracy*). *Precyzja* i *pamięć*

zostały już wyjaśnione wcześniej, natomiast przez *dokładność* należy rozumieć procent poprawnych predykcji. Predykcję międzyprojektową uznawano za wysoce skuteczną jeżeli wszystkie trzy wielkości oceniające model (*precyzja*, *pamięć* i *dokładność*) przyjmowały wartości większe niż $0,75$. Okazało się, że przy tak wysoko postawionej poprzeczce, predykcję międzyprojektową można było uznać za skuteczną jedynie w 21 przypadkach na 622 przebadane, co daje współczynnik sukcesu równy $3,4\%$. Równocześnie autorzy podjęli się identyfikacji czynników, które odgrywały kluczową rolę w sukcesie predykcji międzyprojektowej. Wyniki tej analizy zostały zebrane w postaci drzew decyzyjnych. Osobne drzewo zostało skonstruowane dla każdej z wielkości używanych do oceny predykcji. Niestety autorzy opublikowali jedynie drzewo skonstruowane w oparciu o *precyzję*. Drzewa dla *pamięci* i *dokładności* nie zostały ujawnione. Niemniej jest to eksperyment zdecydowanie najbliższy opisanemu w Rozdziale 5. W związku z tym wyniki eksperymentu przeprowadzonego w laboratoriach MicroSoft Research zostaną szczegółowo porównane z wynikami uzyskanymi w niniejszej pracy i opisane właśnie w Rozdziale 5.

Rozdział 3

Akwizycja danych do badań

3.1 Badane projekty

Wszystkie przeanalizowane projekty to, napisane w całości lub w części w Javie, systemy informatyczne. Analizowano jedną lub kilka wersji wymienionych poniżej projektów.

3.1.1 Projekty otwarte

Największą grupę przebadanych projektów stanowią projekty otwarte (open-source), przebadano ich 15 co przełożyło się na 48 wersji.

Apache Tomcat (http://tomcat.apache.org/). Analizowano wersję 6.0 tego projektu. Apache Tomcat to kontener aplikacji internetowych (ang. web container). Innymi słowy jest to serwer, który umożliwia uruchamianie aplikacji internetowych zrealizowanych w między innymi takich technologiach jak Java Servlets czy też Java Server Pages.

Apache Xalan (http://xml.apache.org/xalan-j). Analizowano cztery wersje tego projektu: 2.4, 2.5, 2.6 oraz 2.7. Projekt ten to procesor języka XSLT, który umożliwia transformowanie dokumentów XML do HTML, plików tekstowych jak i innych dokumentów XML. Implementuje wersję 1.0 *XSL Transofmations* (XSLT) oraz wersję 1.0 *XML Path Language* (XPath). Istnieją wersje projektów Apache Xalan dla różnych języków programowania, tu analizowana była wersja przygotowana dla Javy.

PBeans (http://pbeans.sourceforge.net). Analizowano dwie wersje tego projektu: 1.0 oraz 2.0. PBeans to narzędzie realizujące warstwę dostępu do danych (ang. persistance layer). Zapewnia translację danych pomiędzy relacyjną bazą danych a światem obiektów (ang. O/R mapping). PBeans umożliwia automatyczne wygenerowanie i ewolucję schematu bazy danych na podstawie adnotacji ziaren Javy (ang. Java beans).

Apache Xerces (http://xerces.apache.org/xerces-j). Analizowano cztery wersje tego projektu: 1.1, 1.2, 1.3 oraz 1.4.4. Apache Xerces to pakiet oprogramowania przeznaczony do parsowania i manipulacji dokumentami XML. Implementuje on kilka interfejsów programistycznych dedykowanych do przetwarzania dokumentów XML, takich jak: DOM, SAX, SAX2 i XML Schema 1.0. Istnieją implementacje pakietu Apache Xerces przeznaczone dla języków Java, C++ oraz Perl. Tu analizowano jedynie wersję przeznaczoną dla języka Java.

Apache Ant (http://ant.apache.org). Analizowano pięć wersji tego projektu: 1.3, 1.4, 1.5, 1.6 oraz 1.7. Projekt ten to narzędzie umożliwiające zdefiniowanie zautomatyzowanego procesu budowy oprogramowania. Apache Ant jest w głównej mierze wykorzystywany do budowy projektów rozwijanych w języku Java i obok Apache Maven jest najpopularniejszym narzędziem tego typu.

Apache Ivy (http://ant.apache.org/ivy). Analizowano trzy wersje tego projektu: 1.1, 1.2 oraz 2.0. Apache Ivy to menadżer zależności, który jest zorientowany na elastyczność i prostotę. Umożliwia on zarządzanie zależnościami na poziomie projektu oraz raportowanie tych zależności w postaci graficznej przy pomocy diagramów. Pakiet Apache Ivy jest dostępny zarówno jako samodzielny program jak i jako część składowa Apache Ant.

Apache Camel (http://camel.apache.org). Analizowano cztery wersje tego projektu: 1.0, 1.2, 1.4 oraz 1.6. Apache Camel to szkielet integracyjny. Zawiera on implementacje wielu wzorców integracyjnych, jest prosty w konfiguracji i oferuje wiele gotowych komponentów. Apache Camel dostarcza gotowy silnik routingu oraz mechanizm budowy nowych silników. Umożliwia definiowanie, jakie źródła powinny jakie wiadomości przyjmować, oraz określanie w jaki sposób wiadomości powinny być przetwarzane i przesyłane do kolejnych odbiorców. Apache Camel może być konfigurowany przy pomocy interfejsu programistycznego zdefiniowanego w języku Java, plików XML oraz języka Scala. Współpracuje z wieloma modelami komunikacji, takimi jak: HTTP, JMS, JBI, SCA, Mina czy CXF.

Apache Forrest (http://forrest.apache.org). Analizowano trzy wersje tego projektu: 0.6, 0.7 oraz 0.8. Apache Forrest to narzędzie służące do publikowania treści. Przyjmuje ono na wejściu dane w jednym z wielu formatów i transformuje je do zadanego formatu wyjściowego. Umożliwia również odseparowanie metody prezentacji od prezentowanej treści. Narzędzie to w sporej części zostało napisane w językach skryptowych, takich jak XSLT. Tu analizowano tylko tą jego część, która została stworzona w Javie.

Apache Log4j (http://logging.apache.org/log4j). Analizowano trzy wersje tego projektu: 1.0, 1.1 oraz 1.2. Log4j to biblioteka służąca do tworzenia logów podczas działa aplikacji. Jest konfigurowana przy pomocy zewnętrznych plików. W związku z tym zmiana zachowania mechanizmu logowania programu wykorzystującego *Log4j* nie wymaga ponownej kompilacji.

Apache Lucene (http://lucene.apache.org). Analizowano trzy wersje tego projektu: 2.0, 2.2 oraz 2.4. Apache Lucene to silnik wyszukiwania tekstów wykorzystujący mechanizm indeksowania. Jest skalowalny, ma małe wymagania w stosunku do pamięci operacyjnej. Wspiera używanie rankingów, wyszukiwanie oparte o frazy jak i dżokery (ang. wildcards), sortowanie i modyfikowanie danych podczas wyszukiwania. Istnieją implementacje Apache Lucene wykonane w wielu różnych językach programowania, tu analizowano jednak jedynie wersję przygotowaną w języku Java.

Apache POI (http://poi.apache.org). Analizowano cztery wersje tego projektu: 1.5, 2.0, 2.5.1 oraz 3.0. Projekt ten jest interfejsem programistycznym do dokumentów biurowych w formatach opracowanych przez firmę MicroSoft, takich jak *OLE 2 Compound Document* oraz *OpenXML*. Są to formaty stosowane w popularnych programach biurowych z serii MS Office.

Apache Synapse (http://synapse.apache.org). Analizowano trzy wersje tego projektu: 1.0, 1.1 oraz 1.2. Projekt ten to prosta, lekka i wydajna *Korporacyjna Magistrala Usług* (ang. Enterprise Service

Bus). Apache Synapse wspiera następujące technologie: HTTP, SOAP, SMTP, JMS, FTP, Financial Information eXchange, Hessian, WS-Addressing, Web Services Security, Web Services Reliable Messaging, MTOM oraz XOP.

Apache Velocity (http://velocity.apache.org). Analizowano trzy wersje tego projektu: 1.4, 1.5 oraz 1.6.1. Projekt ten to procesor szablonów stron WWW. Jest on projektowany pod kątem łatwości użycia, a w głównej mierze nakierowany jest na tworzenie dynamicznych stron WWW. Apache Velocity w pełni wspiera architekturę Model-Widok-Kontroler (ang. Model–View–Controler). Pozwala projektantom stron WWW na dołączanie do kodu specyficznych znaczników nazywanych referencjami, dzięki którym można odczytywać i zapisywać wartości atrybutów obiektów z poziomu strony WWW. Apache Velocity jest ciekawą alternatywą dla JSF.

Ckjm (http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm). Analizowano tutaj wersję 1.8 tego narzędzia. Ckjm przetwarza binarny kod Javy w celu wyliczenia metryk oprogramowania. Analizowana wersja tego narzędzia wylicza osiem metryk, w tym wszystkie metryki należące do zestawu CK [13]. Dostępne jest wyjście zarówno w postaci zwykłego pliku tekstowego jak i w formacie XML. Narzędzie integruje się z Apache Ant.

JEdit (http://www.jedit.org). Analizowano pięć wersji tego projektu: 3.2.1, 4.0, 4.1, 4.2 oraz 4.3. JEdit to, dostępny na wiele platform, zaawansowany edytor programisty. Edytor ten oferuje automatyczne formatowanie kodu źródłowego programów, podświetlanie składni, automatyczne uzupełnianie kodu, nagrywanie makr oraz pracę z wyrażeniami regularnymi. Dostępne w JEdit funkcjonalności można rozszerzać przy pomocy wtyczek jak i jednego ze wspieranych przez ten edytor języków skryptowych.

3.1.2 Projekty przemysłowe

Badano sześć projektów przemysłowych. Pięć z nich to kompleksowe systemy bazodanowe o architekturze klient – serwer realizowane na specjalne zamówienie klienta. Dotyczą one branży finansowej. W ramach tych pięciu przemysłowych projektów przebadano łącznie 24 wydania (wersje), każde zakończone wdrożeniem u klienta. Szósty analizowany projekt przemysłowy to rozbudowane narzędzie wspierające proces monitorowania jakości wytwarzanego systemu informatycznego. Przebadano trzy wydania tego narzędzia.

3.1.3 Projekty studenckie

Analizie poddane zostały projekty realizowane przez studentów. Projekty te były realizowane w różnych technologiach, aczkolwiek zawsze z wykorzystaniem języka Java, i dotyczyły bardzo różnych zagadnień biznesowych. Wszystkie były tworzone przez 3 – 5 osobowe grupy studentów czwartego i piątego roku studiów informatycznych. Czas realizacji projektów wynosił dwa semestry. Łącznie przebadano siedemnaście projektów studenckich, każdy z nich miał dokładnie jedno wydanie.

3.2 Charakterystyka wykorzystanych metryk produktu

W rozdziale podano statystyki dotyczące wykorzystanych metryk produktu. Statystyki wyliczono na podstawie wszystkich badanych projektów. W Tabeli 3.1 zaprezentowano wartości wyliczone dla wszystkich projektów łącznie. Natomiast na wykresach, oraz w Załączniku A dla każdego projektu z osobna. Wyliczenia przeprowadzone dla każdego z projektów z osobna są wyjątkowo ciekawe w świetle eksperymentów dotyczących predykcji międzyprojektowej. Jeżeli jakaś metryka w jednym z projektów

jest dodatnio skorelowana z liczbą defektów, a w innym ujemnie, to można podejrzewać, że predykcja między tymi dwoma projektami prowadzona w oparciu o tą metrykę nie zakończy się sukcesem.

Tabela 3.1: Charakterystyki metryk produktu

Metryka	Średnia	Minimum	Maksimum	σ	r	r_s
WMC	6,33	0	413	10,64	,24	0,14
DIT	2,84	1	9	1,70	-0,07	-0,08
NOC	,58	0	546	7,93	0,01	0,05
CBO	14,11	0	860	20,15	0,20	0,10
RFC	25,38	0	583	29,51	0,31	0,17
LCOM	51,69	0	59977	678,28	0,10	0,09
LCOM3	,31	0	2	0,59	-0,08	-0,09
NPM	4,43	0	347	9,04	0,20	0,16
DAM	,26	0	1	0,40	0,10	0,11
MOA	,23	0	158	1,29	0,19	0,16
MFA	,58	0	1	0,41	-0,07	-0,07
CAM	,54	0	1	0,24	-0,16	-0,17
IC	,97	0	6	1,05	-0,05	-0,06
CBM	1,66	0	33	2,52	0,02	-0,03
AMC	29,81	0	3492	49,11	0,10	0,14
Ca	3,32	0	860	17,25	0,15	0,09
Ce	10,91	0	133	10,18	0,16	0,07
Max(CC)	3,46	0	252	6,22	0,18	0,14
Avg(CC)	1,29	0	31	1,43	0,11	0,13
LOC	195,95	0	23683	461,04	0,29	0,21

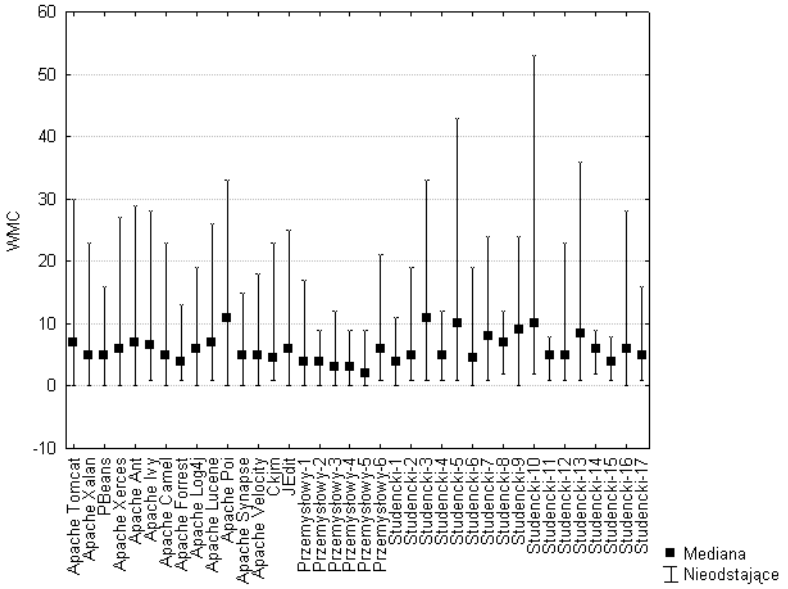
Na umieszczonych w tym rozdziale prezentowano rozrzut wartości przyjmowanych przez metryki w poszczególnych projektach opierając się na wartości mediany. W Tabeli 3.1 natomiast podano wartości średnie, minimalne i maksymalne oraz odchylenie standardowe (σ). Wyliczono również korelację metryki z liczbą defektów; wyliczono współczynnik korelacji liniowej Pearsona (r) oraz współczynnik korelacji rang Spearmana (r_s).

Weighted Methods per Class (WMC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna znajdują się w załączniku w Tabeli A.1 oraz na Rysunku 3.1. Na podstawie tych danych można stwierdzić, że wartość tej metryki w umiarkowanym stopniu zmienia się pomiędzy projektami, ale za to jest różnorodna w obrębie poszczególnych projektów. Zarówno średnia jak i odchylenie standardowe oscylują w okolicach wartości 10. W przypadku większości projektów metryka ta jest mocno skorelowana z liczbą defektów, w przypadku dwóch projektów (*Ckjm* i *Studencki-17*) przekracza wartość 0,8.

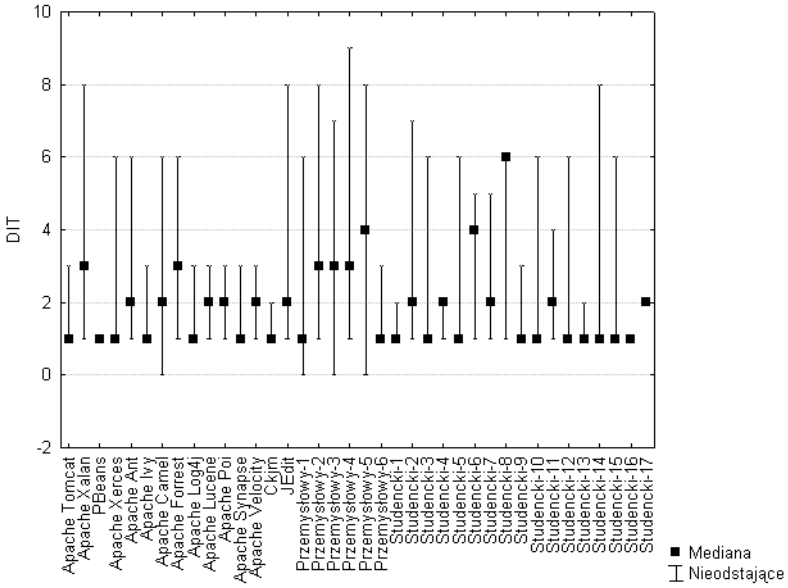
Depth of Inheritance Tree (DIT). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.2 oraz na Rysunku 3.2. Na podstawie tych danych można stwierdzić, że dziedziczenie jest używane w badanych projektach w ograniczonym stopniu. Wartość średnia tej metryki, podobnie jak mediana tyko w przypadku nielicznych projektów przekracza wartość 3. Metryka ta jest również słabo skorelowana z liczbą defektów. W przypadku wielu projektów korelacja przyjmuje wartości ujemne, ale bliskie 0.

Number Of Children (NOC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.3 oraz na Rysunku 3.3. Wartości przyjmowane przez tą metrykę potwierdzają obserwację poczynioną przy okazji metryki DIT, mianowicie ograniczone wykorzystywanie mechanizmu dziedziczenia. Metryka ta jest słabo skorelowana z liczbą defektów. Jak można zobaczyć w Tabeli 3.1 uzyskano współczynniki korelacji równe odpowiednio $0,01$ oraz $0,05$.

Coupling Between Object classes (CBO). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.4 oraz na Rysunku 3.4.

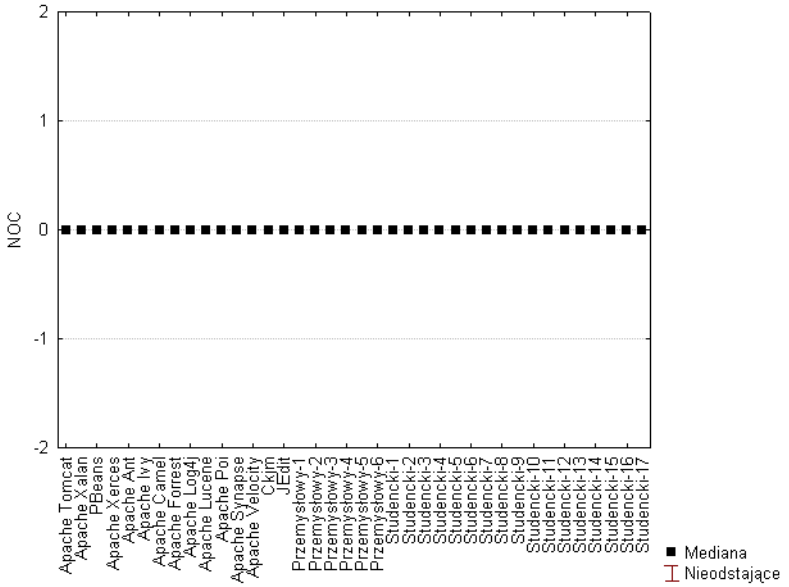


Rysunek 3.1: Weigthed Methods per Class

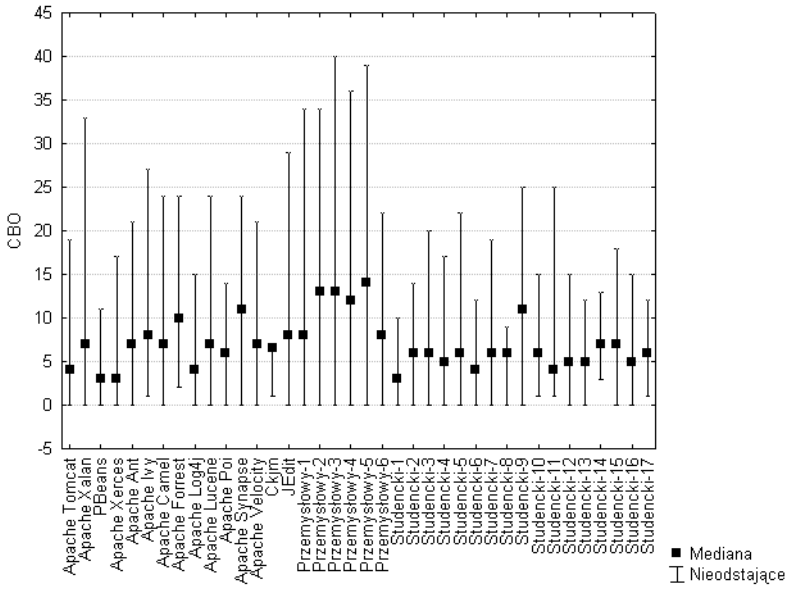


Rysunek 3.2: Depth of Inheritance Tree

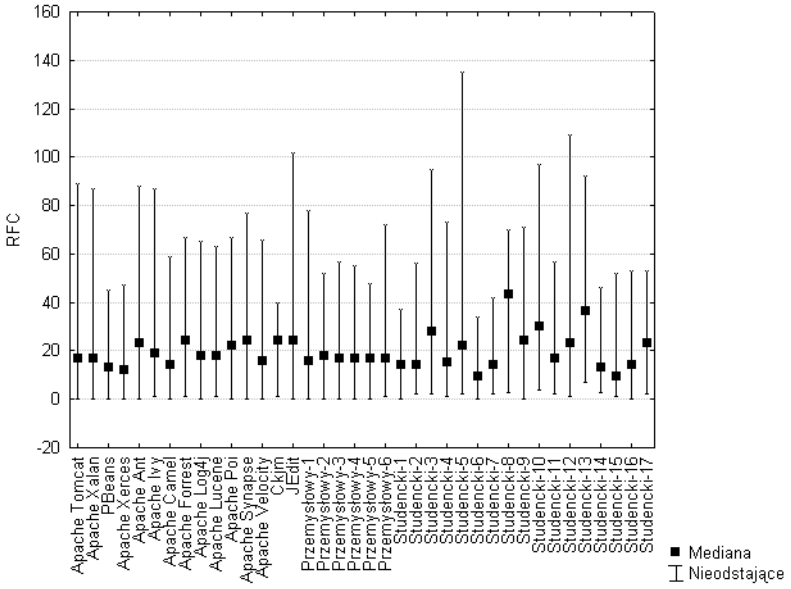
Zaprezentowane dane pozwalają na stwierdzenie, że najwyższe wartości (średnie i mediany) przyjmuje ta metryka w projektach przemysłowych. Metryka ta jest dosyć mocno skorelowana z liczbą defektów. Najwyższa korelacja występuje w projekcie *Studencki-5*: $r = 0,73$, $r_s = 0,74$.



Rysunek 3.3: Number Of Children

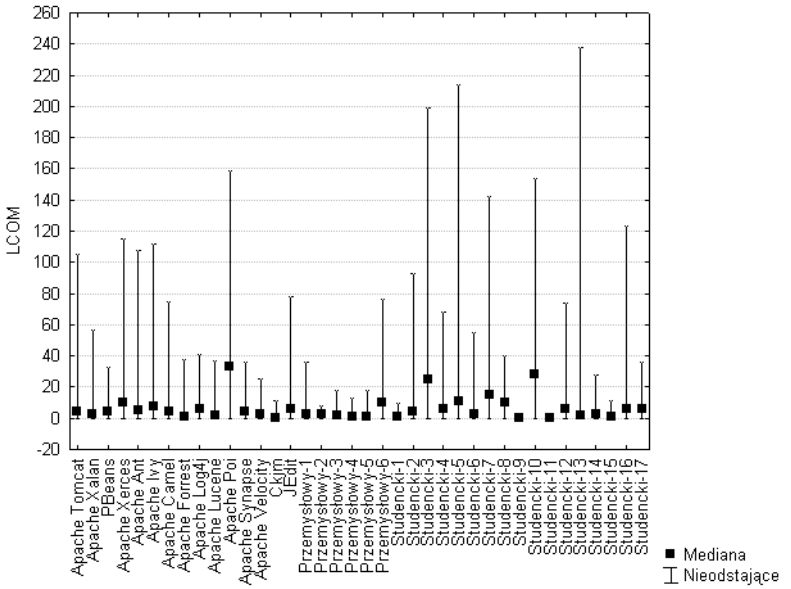


Rysunek 3.4: Coupling Between Object classes



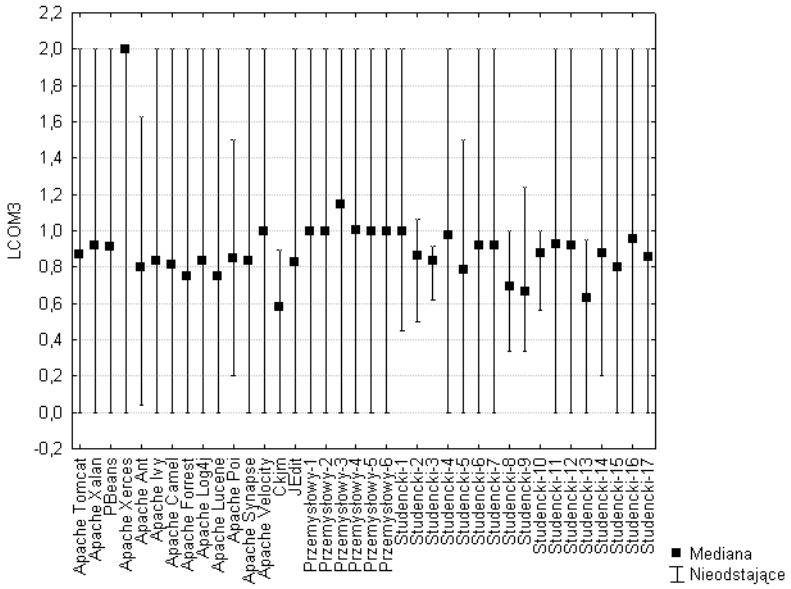
Rysunek 3.5: Response For a Class

Response For a Class (RFC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.5 oraz na Rysunku 3.5. Wartości przyjmowane przez tą metrykę cechują się dużym rozrzutem zarówno pomiędzy projektami, jak i w obrębie poszczególnych projektów. Wartość średnia przyjmuje wartość od 12,59 do 57,93, natomiast odchylenie standardowe w skrajnym przypadku (projekt *Studencki-5*) wynosi 76,87. Korelacja tej metryki z liczbą defektów w zdecydowanej większość projektów jest wysoka. Współczynnik korelacji Pearsona wyliczony dla wszystkich projektów łącznie jest najwyższy w przypadku tej właśnie metryki i wynosi 0,31 (Tabela 3.1).



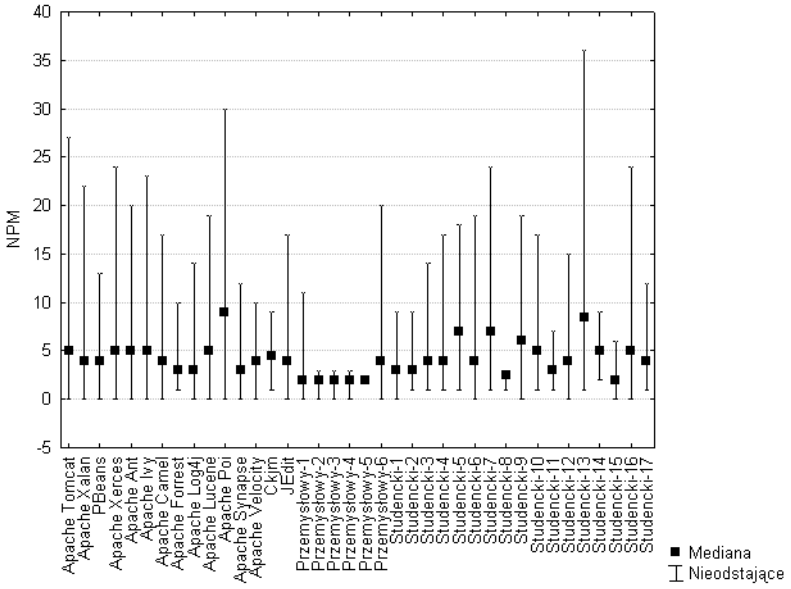
Rysunek 3.6: Lack of Cohesion in Methods

Lack of Cohesion in Methods (LCOM). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.6 oraz na Rysunku 3.6. Wartości średnie przyjmowane przez tą metrykę bardzo różnią się w przypadku poszczególnych projektów. Dla projektów *Studencki-9* i *Apache Forrest* jest to odpowiednio 6,55 i 11,72, kiedy dla projektu *Studencki-5* jest to aż 260,4. Zmienność tej metryki w obrębie poszczególnych projektów jest również spora – odchylenie standardowe dla projektu JEdit wynosi aż 1804,04. W przypadku różnych projektów metryka ta jest skorelowana z liczbą defektów w bardzo różnym stopniu. Współczynnik korelacji Pearsona waha się od wartości -0,37 do 0,93.



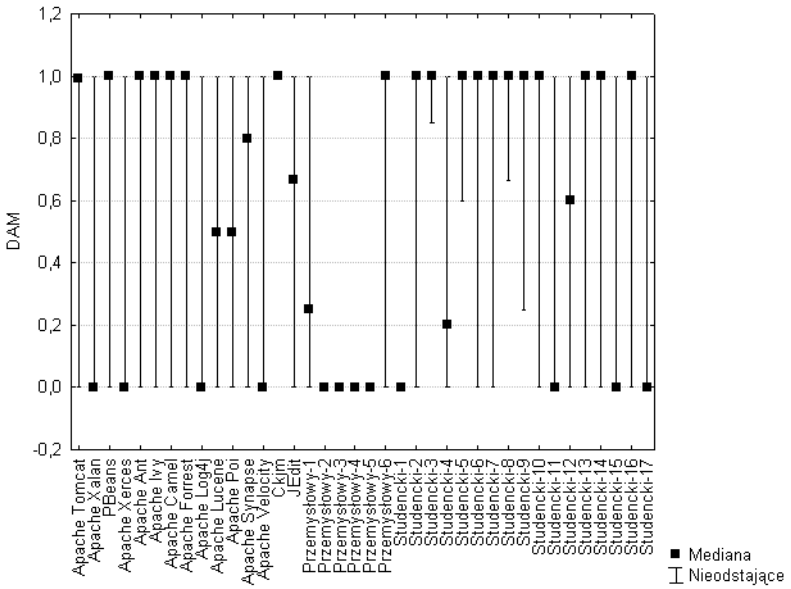
Rysunek 3.7: Lack of Cohesion in Methods (3)

Lack of Cohesion in Methods (LCOM3). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.7 oraz na Rysunku 3.7. Interpretując wartości przyjmowane przez tą metrykę należy zwrócić uwagę na fakt, że metryka ta może przyjmować tylko wartości z przedziału od 0 do 2. W przypadku tej metryki zastanawiający jest fakt, że metryka ta bardzo często przyjmuje wartości przekraczając 1, a przyjęcie takiej wartości wskazuje na poważny brak spójności klasy. Metryka ta jest bardzo słabo skorelowana z liczbą defektów. W przypadku zdecydowanej większości projektów współczynnik korelacji jest minimalnie mniejszy od 0.



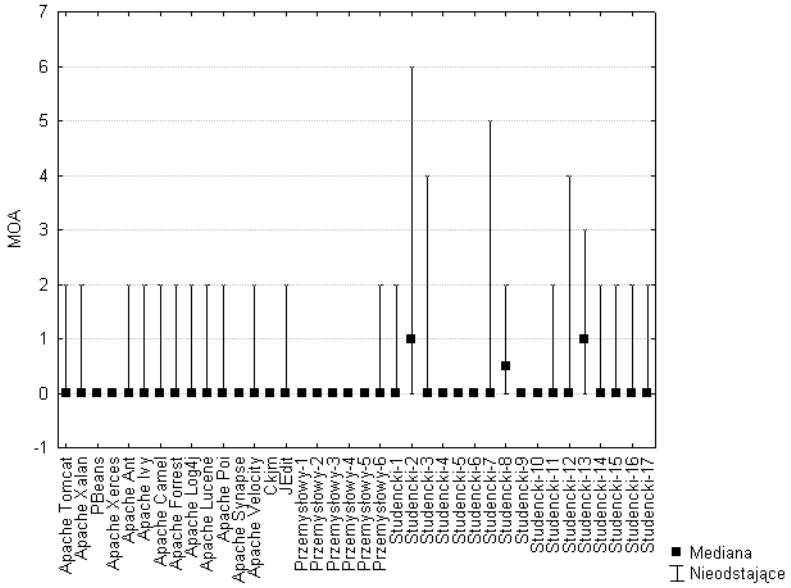
Rysunek 3.8: Number of Public Methods

Number of Public Methods (NPM). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.8 oraz na Rysunku 3.8. Wartość tej metryki w umiarkowanym stopniu zmienia się pomiędzy projektami, ale za to jest różnorodna w obrębie poszczególnych projektów. Zarówno średnia jak i odchylenie standardowe oscylują w okolicach wartości 10. W przypadku większości projektów metryka ta jest mocno skorelowana z liczbą defektów, w przypadku jednego projektu (*Ckjm*) przekracza wartość 0,8. Obserwacje te wskazują na bardzo duże podobieństwo do metryki WMC.



Rysunek 3.9: Data Access Metric

Data Access Metric (DAM). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.9 oraz na Rysunku 3.9. Interpretując wartości przyjmowane przez tą metrykę należy zwrócić uwagę na fakt, że metryka ta może przyjmować tylko wartości z przedziału od 0 do 1. W świetle tego ograniczenia można stwierdzić, że wartość metryki jest bardzo zmienna. Jej wartość średnia waha się od 0,07 w projektach *Przemysłowy-5* i *Studencki-1* do 0,89 w projekcie *Ckjm*, a mediana od 0 do 1 (wartości skrajne mediany występują w kilku projektach). Odchylenie standardowe w większości przypadków jest bliskie wartości 0,5. Korelacja metryki DAM z liczbą defektów jest w przeważającej liczbie projektów niewielka, największa jest dla projektu *Studencki-13* i wynosi $r_s = 0,41$.

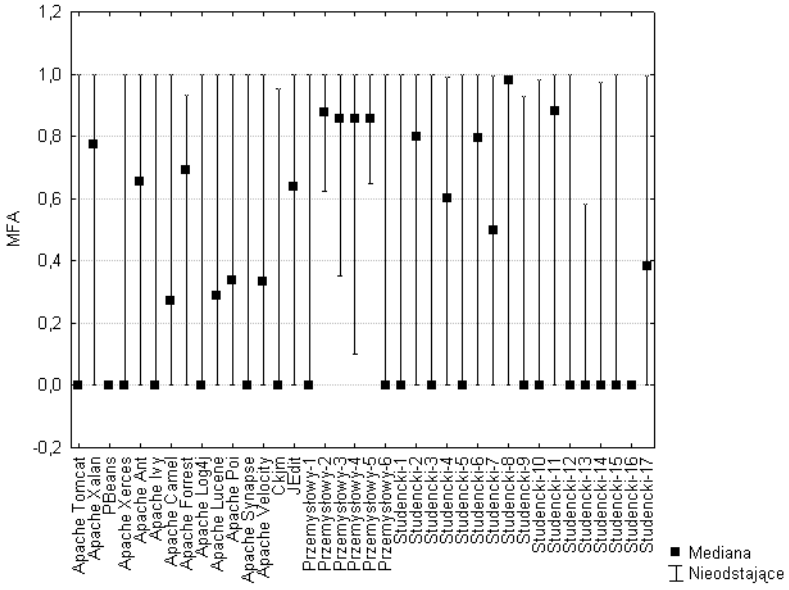


Rysunek 3.10: Measure Of Aggregation

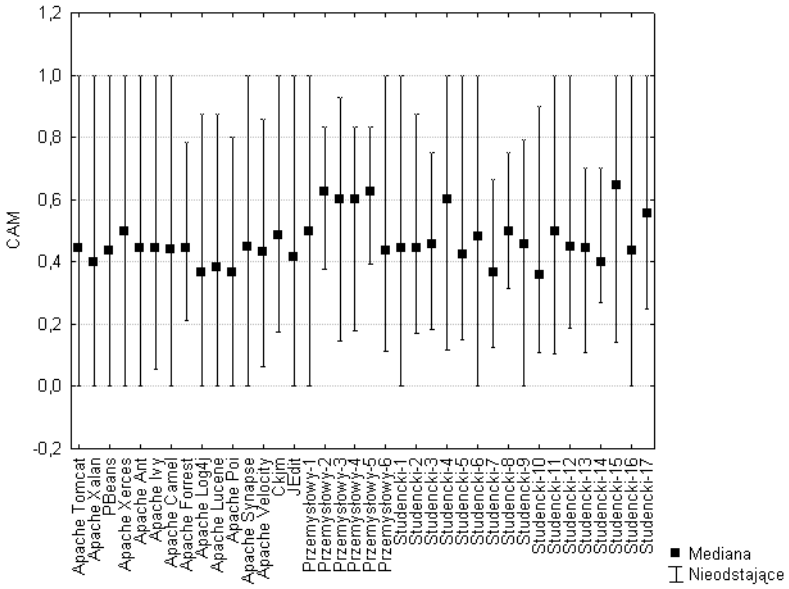
Measure Of Aggregation (MOA). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.10 oraz na Rysunku 3.10. Na podstawie tych danych można stwierdzić, że wartość tej metryki jest w większości klas badanych projektów bardzo mała. Mediana w przypadku niemal wszystkich projektów jest równa 0, a wartość średnia rzadko przekracza 1. Korelacja z liczbą defektów przyjmuje bardzo różne wartości. W przypadku kilku projektów jest ona ujemna, ale są również projekty, w których przekracza ona wartość 0,6.

Measure of Functional Abstraction (MFA). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.11 oraz na Rysunku 3.11. Interpretując wartości przyjmowane przez tą metrykę należy zwrócić uwagę na fakt, że metryka ta może przyjmować tylko wartości z przedziału od 0 do 1. W świetle tego ograniczenia można stwierdzić, że metryka ta przyjmuje różnorodne wartości zarówno pomiędzy projektami jak i w obrębie poszczególnych projektów. Wskazują na to zarówno wartości średnie, mediany jak i odchylenia standardowe. Korelacja tej metryki z liczbą defektów jest bliska zeru w przypadku projektów otwartych i przemysłowych. W przypadku projektów studenckich korelacje są o wiele bardziej różnorodne, w większości przypadków są dodatnie.

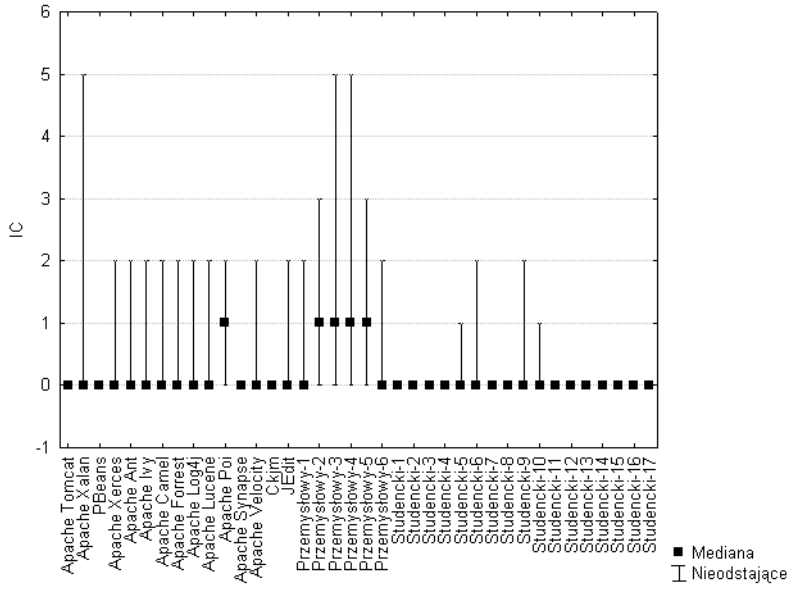
Cohesion Among Methods of class (CAM). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.12 oraz na Rysunku 3.12. Interpretując wartości przyjmowane przez tą metrykę należy zwrócić uwagę na fakt, że metryka ta może przyjmować tylko wartości z przedziału od 0 do 1. Wartość średnia tej metryki, podobnie jak mediana, w większości przypadków jest bliska wartości 0,5. Stosunkowo małe jest rozproszenie wartości przyjmowanych przez tą metrykę w obrębie poszczególnych projektów. Odchylenie standardowe zazwyczaj jest bliskie wartości 0,2. Metryka CAM jest ujemnie skorelowana z liczbą defektów we wszystkich projektach za wyjątkiem dwóch studenckich.



Rysunek 3.11: Measure of Functional Abstraction



Rysunek 3.12: Cohesion Among Methods of class



Rysunek 3.13: Inheritance Coupling

Inheritance Coupling (IC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.13 oraz na Rysunku 3.13. Wartości średnie metryki IC w większości projektów są stosunkowo niskie, a mediany w prawie wszystkich przypadkach są równe 0. Zestawiając to z przedstawionymi w Tabeli A.13 odchyleniami standardowymi można wyciągnąć wniosek, że w przypadku przeważającej liczby klas wartość tej metryki wynosi 0, ale równocześnie istnieją klasy, dla których metryka ta przyjmuje zdecydowanie większe wartości. Metryka ta przyjmuje zdecydowanie największe wartości dla projektów przemysłowych. Metryka IC wydaje się być słabym predykatorem defektów, ponieważ jej korelacje z liczbą defektów są bliskie 0 (czasem dodatnie, czasami ujemne).

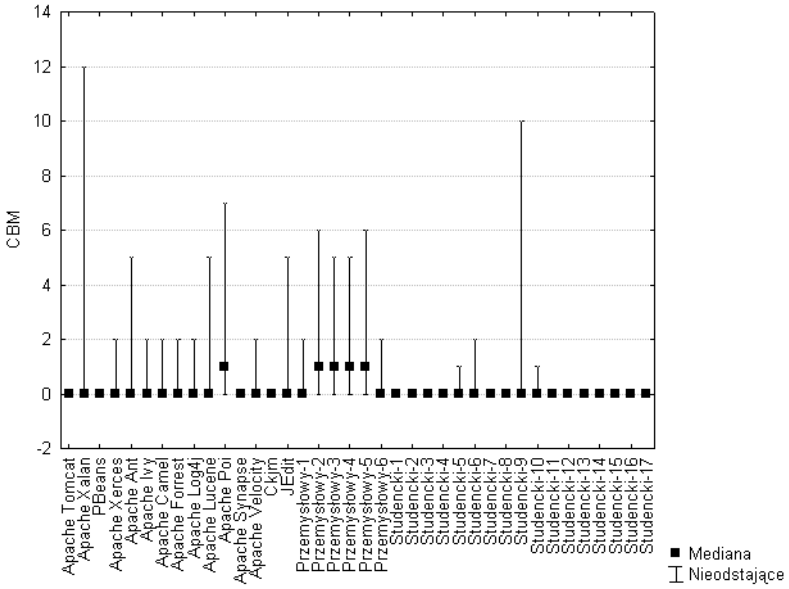
Coupling Between Methods (CBM). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowano w załączniku w Tabeli A.14 oraz na Rysunku 3.14. Metryka CBM cechuje się odrobinę większą zmiennością niż metryka IC, ale podobnie jak IC wydaje się być słabym predykatorem defektów. Jej korelacja z liczbą defektów przyjmuje wartości od -0,31 do 0,45 w przypadku korelacji Pearsona oraz od -0,36 do 0,44 w przypadku korelacji rang Spearmana.

Average Method Complexity (AMC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.15 oraz na Rysunku 3.15. Wartości średnie tej metryki są bardzo różnorodne. Najmniejsza to zaledwie 5,96 w przypadku projektu *Studencki-7*, a największa to aż 57,36, wyliczona dla projektu *Apache Xalan*. Metryka AMC jest pozytywnie skorelowana z liczbą defektów w przypadku wszystkich projektów za wyjątkiem jednego (*Studencki-13*), ale zazwyczaj wartość tej korelacji nie jest zbyt wysoka.

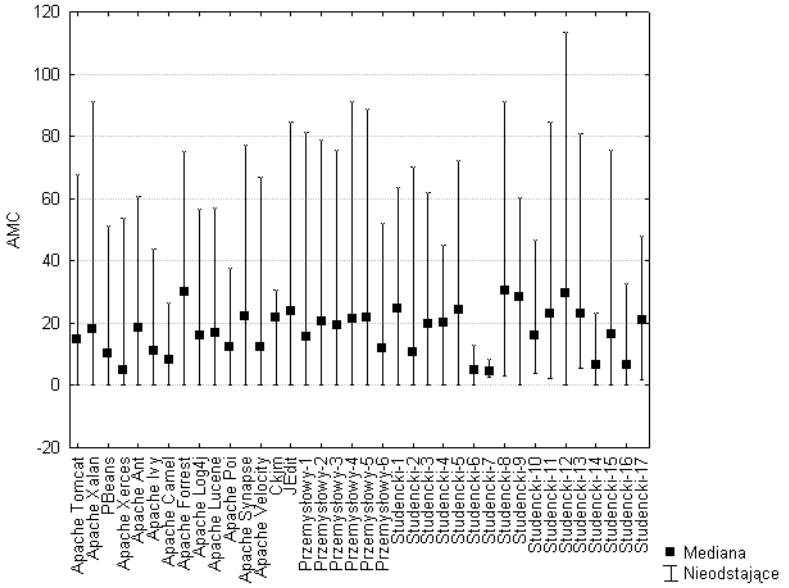
Afferent coupling (Ca). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.16 oraz na Rysunku 3.16. W charakterystyce metryki Ca najbardziej rzuca się w oczy ogromny rozrzut wartości maksymalnych przyjmowanych przez tą metrykę w poszczególnych projektach. Istnieją projekty, dla których jest ona mniejsza od 10,

ale są również takie, dla których przekracza ona wartość 500. W przypadku wielu projektów metryka ta jest mocno skorelowana z liczbą defektów. Istnieją jednak również takie projekty, dla których ta korelacja jest ujemna. Sugeruje to, że metryka Ca tylko w niektórych projektach będzie użyteczna w modelach predykcji defektów.

Efferent couplings (Ce). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna podano w załączniku w Tabeli A.17 oraz na Rysunku 3.17. Metryka ta przyjmuje zdecydowanie największe wartości dla projektów przemysłowych. Obserwację tą potwierdzają zarówno wartości median przedstawione na Rysunku A.17, jak i zaprezentowane w Tabeli A.17 wartości średnie.



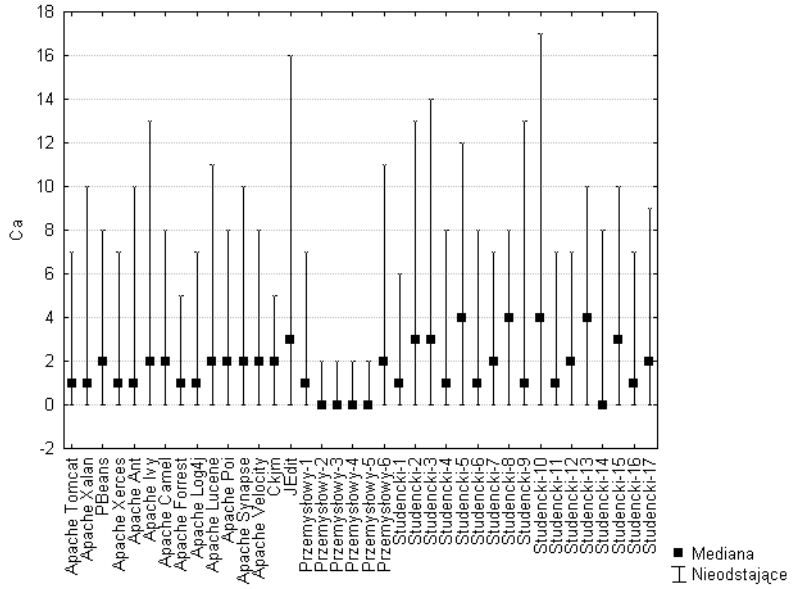
Rysunek 3.14: Coupling Between Methods



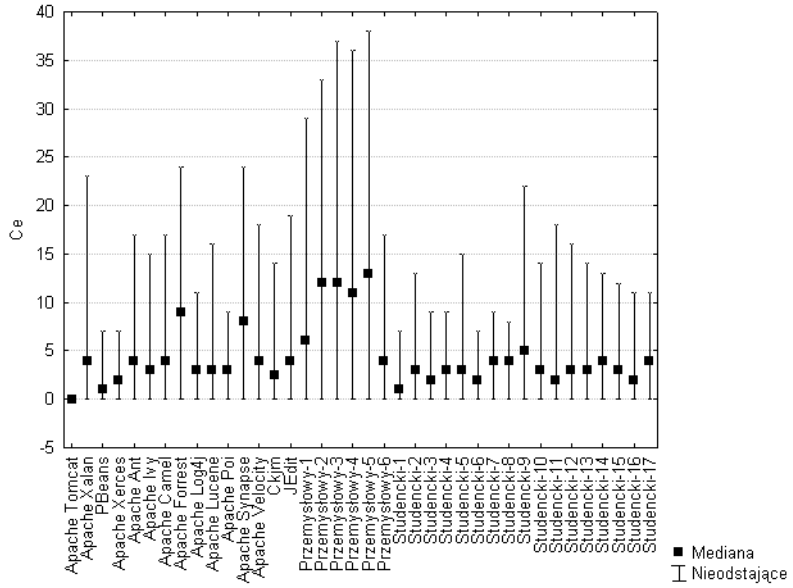
Rysunek 3.15: Average Method Complexity

Metryka Ce jest skorelowana z liczbą defektów. Z najwyższą korelacją mamy do czynienia w przypadku projektu *Studencki-5*: $r = 0,75$, $r_s = 0,70$.

McCabe's Cyclomatic Complexity (CC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna przedstawione zostały w załączniku w Tabeli A.18 oraz na Rysunku 3.18 dla metryki Max(CC) oraz w załączniku w Tabeli A.19 oraz na Rysunku 3.19 dla metryki Avg(CC). Zgodnie z tym co bezpośrednio wynika z definicji metryk Max(CC) oraz Avg(CC), pierwsza z nich przyjmuje zdecydowanie większe wartości. Zróżnicowanie wartości przyjmowanych przez te metryki w poszczególnych projektach jest stosunkowo niewielkie. Mediany są bliskie 1 w przypadku wszystkich



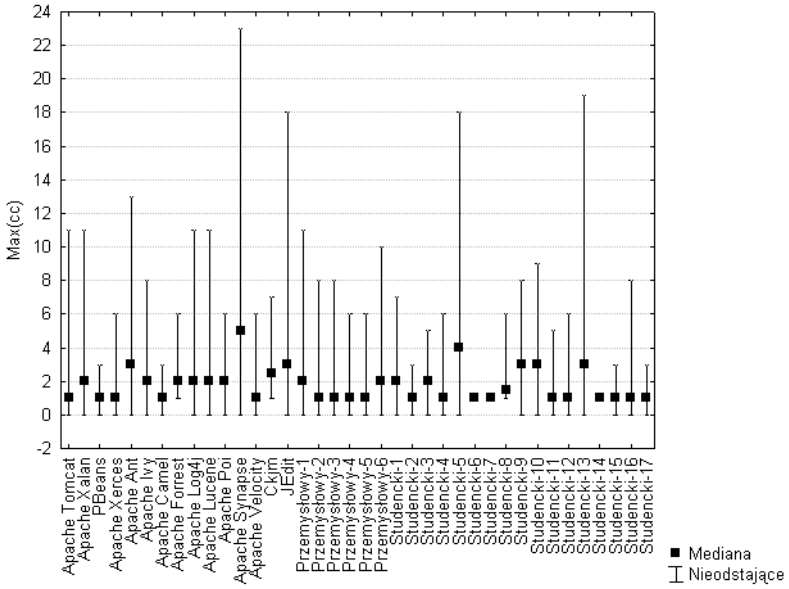
Rysunek 3.16: Afferent coupling



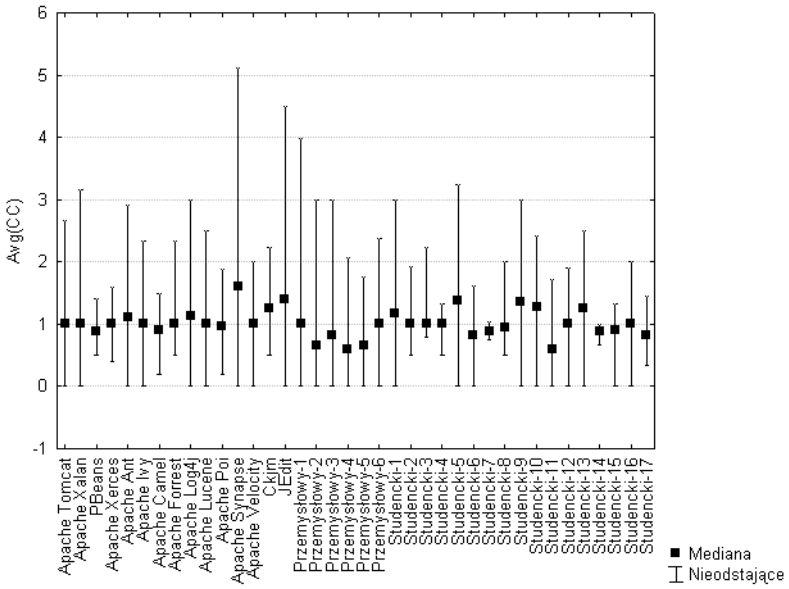
Rysunek 3.17: Efferent couplings

projektów, a wartości średnie mieszczą się w przedziale od $0,73$ do $2,01$. Może to wskazywać na bardzo intensywne używanie akcesorów, które zazwyczaj są metodami o bardzo małej złożoności cyklomatycznej. Korelacje z liczbą defektów są w większości przypadków niezbyt wysokie, chociaż istnieje kilka projektów, w których daje się zauważyć powiązanie pomiędzy defektami a złożonością cyklomatyczną. W przypadku metryki Max(CC) na przykład w projekcie *Studencki-2*: $r = 0,62$, $r_s = 0,23$; a w przypadku metryki Avg(CC) w projekcie *Studencki-4*: $r = 0,58$, $r_s = 0,29$.

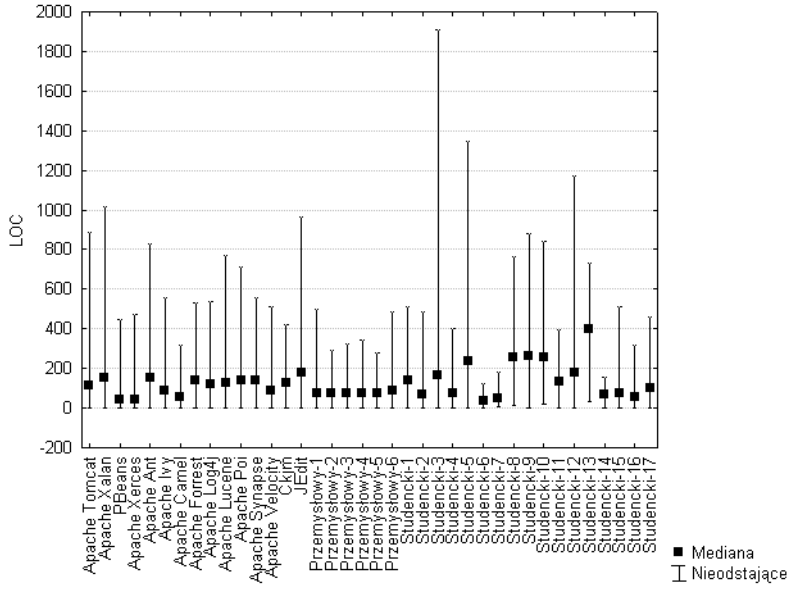
Lines of Code (LOC). Statystyki opisowe oraz korelacje z liczbą defektów wyliczone dla każdego projektu z osobna zaprezentowane zostały w załączniku w Tabeli A.20 oraz na Rysunku 3.20. Wartości



Rysunek 3.18: Maksymalna złożoność cyklomatyczna



Rysunek 3.19: Średnia złożoność cyklomatyczna



Rysunek 3.20: Lines of Code

przyjmowane przez metrykę LOC są bardzo różnorodne. Wskazują na to zarówno wartości średnie i odchylenia standardowe przedstawione w Tabeli A.20 jak i mediany zaprezentowane na Rysunku 3.20. Metryka ta wydaje się być dobrym predykatorem defektów, jej korelacje z liczbą defektów są w większości projektów wysokie i oscylują w okolicach wartości $0,5$. Najwyższe wartości korelacji uzyskano dla projektu *Studencki-5*: $r = 0,74$, $r_s = 0,81$

3.3 Charakterystyka badanych metryk procesu

Tabela 3.2: Charakterystyka metryk procesu

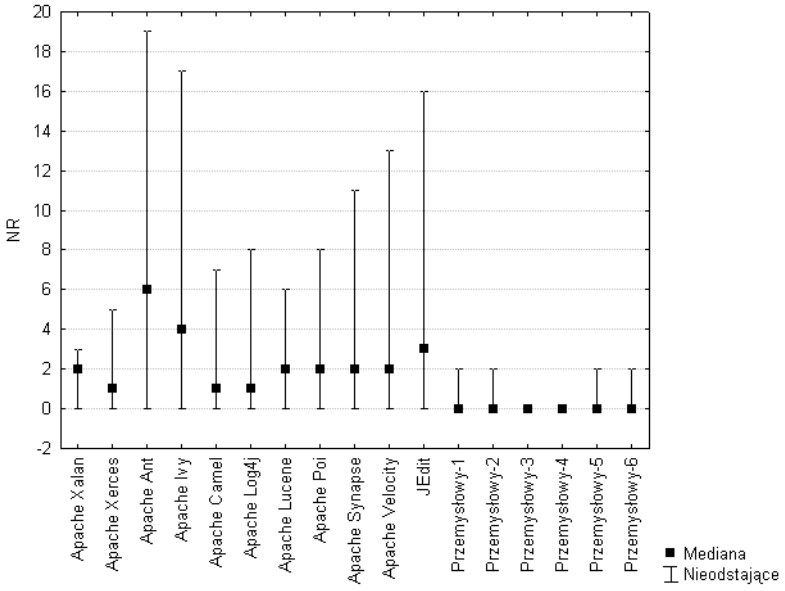
Metryka	Średnia	Minimum	Maksimum	σ	r	r_s
NR	1,27	0	169	3,54	0,31	0,28
NDC	0,68	0	16	1,19	0,27	0,27
NML	19,77	0	5584	113,65	0,27	0,20
NDPV	0,26	0	47	1,02	0,36	0,24
NER	0,32	0	21	0,82	0,22	0,12
NPR	0,14	0	7	0,43	0,11	-0,01

Tabela 3.3: Charakterystyka metryki IN

Liczba starych klas	Liczba nowych klas	Stare klasy z defektami [%]	Nowe klasy z defaktami [%]
60019	9722	16,04	2,15

W rozdziale tym zostały opisane wszystkie metryki procesu, które były badane pod kątem ich przydatności w modelach predykcji defektów. Charakterystyki tych metryk zostały zebrane w tabelach 3.2 i 3.3. Tabele te zawierają dane zbiorcze, dotyczące wszystkich przebadanych projektów. Niestety w przypadku niektórych metryk nie udało się zebrać danych we wszystkich wymienionych w Rozdziale

3.1 projektach. W związku z tym część danych przedstawionych w tabelach 3.2 i 3.3 dotyczy jedynie podzbioru tych projektów. Szczegółowe dane o metrykach procesu, dotyczące każdego z badanych projektów z osobna, znajdują się w Załączniku B.



Rysunek 3.21: Number of Revisions

Number of Revisions (NR). Przydatność metryki NR w predykcji defektów została poddana analizie z uwagi na często przytaczaną w badaniach nad defektami tezę mówiącą, że każda modyfikacja kodu źródłowego jest potencjalnym źródłem defektów. W związku z tym możliwe jest, że klasy częściej modyfikowane kryją w sobie większą liczbę defektów. Szczegóły dotyczące metryki NR, w tym lista projektów, dla których ta metryka została wyliczona, znajdują się na Rysunku 3.21 oraz w załączniku w Tabeli B.1. Na podstawie tych danych można stwierdzić, że istnieje spora różnica pomiędzy projektami przemysłowymi a otwartymi – w tych drugich dokonuje się zdecydowanie więcej modyfikacji. Korelacje metryki NR z liczbą defektów są dosyć wysokie i niemal we wszystkich przypadkach dodatnie. Dochodzą do wartości 0,67 w przypadku korelacji Pearsona dla projektu *Apache Lucene* oraz do wartości 0,52 w przypadku korelacji rang Spearmana dla projektu *Apache Camel*.

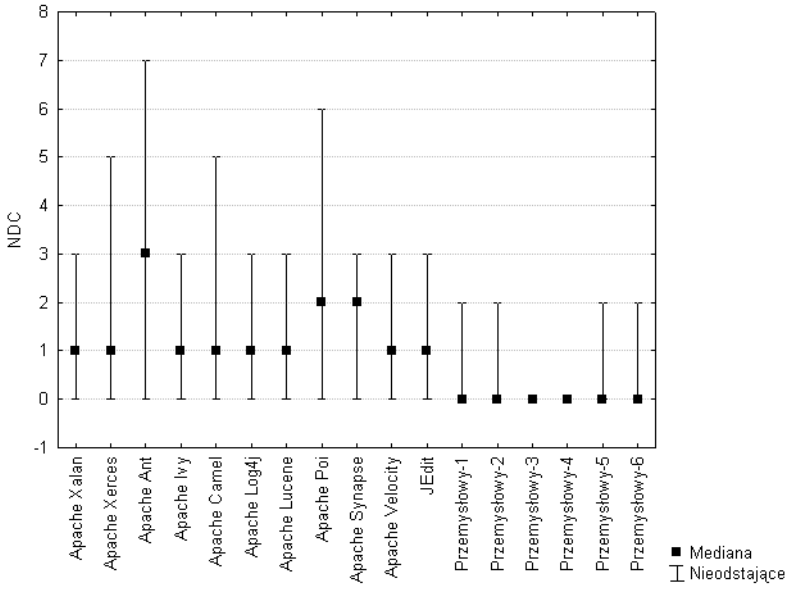
Number of Distinct Commiters (NDC). Motywacją stojącą za metryką NDC jest zwiększanie się możliwości powstawania chaosu w klasach modyfikowanych przez wielu różnych programistów. Programiści ci mogą nie rozumieć zmian wykonywanych przez swoich kolegów albo w nieodpowiedzialny sposób je usuwać. Szczegóły dotyczące metryki NDC, w tym lista projektów, dla których ta metryka została wyliczona, znajdują się na Rysunku 3.22 oraz w załączniku w Tabeli B.2. Charakterystyka tej metryki jest zbliżona do metryki NR. Metryka NDC również przyjmuje zdecydowanie niższe wartości dla projektów przymysłowych niż dla otwartych. Jest jednak, w odróżnieniu od NR, niezbyt mocno skorelowana z liczbą defektów. Zarówno korelacja Pearsona, jak i korelacja rang Spearmana jest w przypadku większości projektów bliska wartości 0,25.

Number of Modified Lines (NML). Motywacja stojąca za tą metryką jest podobna do motywacji kryjącej się za metryką NR. Każda modyfikacja kodu może być źródłem błędów. Możliwe jest jednak, że znaczenie ma nie tylko liczba modyfikacji, ale również ich rozmiar. Szczegóły dotyczące metryki

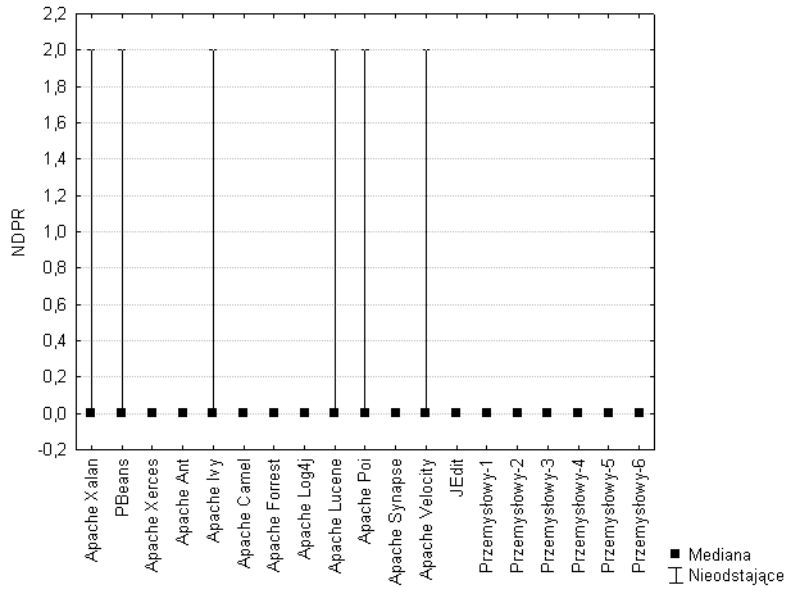
NML, w tym lista projektów, dla których ta metryka została wyliczona, znajdują się w załączniku w Tabeli B.3. Rysunek z wartościami mediany dla tej metryki nie został zaprezentowany, gdyż metrykę tą wyliczono tylko dla projektów przemysłowych. Przebadane projekty przemysłowe cechują się niezbyt dużą intensywnością modyfikacji w efekcie czego mediana z metryki NML była równa zero we wszystkich przypadkach. Metryka NML jest w umiarkowanym stopniu skorelowana z liczbą defektów. Korelacja Pearsona jest na zbliżonym poziomie w przypadku wszystkich przeanalizowanych projektów i zmienia się od 0,24 do 0,36. Nie jest tak jednak w przypadku korelacji rang Spearmana. Istnieje jeden projekt dla którego r_s jest jedynie minimalnie mniejsze od 0. W pozostałych projektach korelacja rang Spearmana zmienia się od 0,13 do 0,38.

Is New (IN). Klasy istniejące w systemie od dłuższego czasu powinny być już klasami dojrzałymi. Można w związku z tym podejrzewać, że w klasach tych będzie się znajdować zdecydowanie mniej błędów niż w nowych klasach. W związku z tym wydaje się być uzasadnione przebadanie metryki IN pod kątem możliwości stosowania jej do predykcji liczby defektów. Szczegóły dotyczące metryki IN, w tym lista projektów, dla których ta metryka została wyliczona, znajdują się w załączniku w Tabeli B.4. Obserwacje empiryczne okazują się jednak być sprzeczne z podaną motywacją stojącą za metryką IN. Częstość występowania defektów w nowych klasach jest zdecydowanie niższa niż w starych klasach w większości przebadanych projektów. Wśród analizowanych projektów istnieje tylko jeden (*Apache Ivy*), w którym w nowych klasach jest procentowo więcej defektów niż w starych.

Number of Defects in Previous Version (NDPV). Motywacja stojąca za tą klasą jest stosunkowo prosta. Jeżeli z jakąś klasą były problemy w przeszłości to można podejrzewać, że będą z nią też problemy w przyszłości. Szczegóły dotyczące metryki NDPV, w tym lista projektów, dla których ta metryka została wyliczona, znajdują się na Rysunku 3.23 oraz w załączniku w Tabeli B.5. Metryka ta przyjmuje niezbyt wysokie wartości, oprócz median potwierdzają to również wartości średnie, ale mimo to wydaje się być całkiem dobrym predykatorem liczby defektów. Jej korelacja z liczbą defektów naprawionych po chwili czasowej, w której metrykę mierzono (czyli w ramach następnego wydania projektu), jest wysoka w większości przebadanych projektów. Dla *JEdit* jest to aż $r = 0,74$ i $r_s = 0,45$.



Rysunek 3.22: Number of Distinct Commiters



Rysunek 3.23: Number of Defects in Previous Version

Number of Evening Revisions (NER). Motywacją stojącą za tą metryką jest presja jaką wywołuje zbliżający się koniec dnia roboczego. Programista wiedząc, że pozostaje mu niewiele czasu do wyjścia z pracy, może zatwierdzić modyfikacje, których mając więcej dostępnego czasu by nie zaakceptował. Szczegóły dotyczące metryki NER znajdują się w załączniku w Tabeli B.6. Metryka ta była badana tylko w jednym projekcie w związku z tym w Tabeli B.6 podano jej charakterystykę w poszczególnych wersjach tego projektu. Nie podano rysunku opartego o mediany metryki NER, ponieważ we wszystkich przebadanych wersjach projektu mediana była równa zero. Metryka ta przyjmuje niezbyt wysokie wartości. Wyliczone korelacje z liczbą defektów również nie wyglądają nazbyt zachęcająco, najwyższa uzyskana wartość to 0,38.

Number of Pre-code-freeze Revisions (NPR). Jest to druga z kolei metryka mająca za zadanie unaocznić wykonywanie zadań programistycznych pod presją. W przypadku tej metryki presja wynika ze zbliżającego się terminu zakończenia prac związanych z przygotowywaniem kolejnego wydania rozwijanego systemu. Szczegóły dotyczące metryki NPR znajdują się w załączniku w Tabeli B.7. Metryka ta była badana tylko w jednym projekcie w związku z tym w Tabeli B.7 podano jej charakterystykę w poszczególnych wersjach tego projektu. Nie podano rysunku opartego o mediany metryki NPR, ponieważ we wszystkich przebadanych wersjach projektu mediana była równa zero. Metryka ta przyjmuje jeszcze mniejsze wartości niż opisana powyżej NER. Metrykę tą analizowano w zaledwie dwóch wersjach jednego projektu. Zebrane dane mogą być zatem niewystarczające i niemiarodajne. Niemniej jednak, nie dają one mocnych przesłanek ku temu, że metryka ta może być dobrym predykatorem liczby defektów – wyliczone korelacje są stosunkowo niskie.

3.4 Narzędzia

W ramach prac związanych z wyliczaniem, gromadzeniem oraz zarządzaniem metrykami powstało kilka narzędzi, które zostały następnie opublikowane w internecie i mogą być wykorzystywane zarówno w dalszych badaniach jak i w zarządzaniu jakością rzeczywistych projektów programistycznych.

3.4.1 Ckjm

Ckjm to narzędzie wyliczające metryki produktu z kodu binarnego Javy. W początkowym zamyśle narzędzie to miało wyliczać sześć metryk z zestawu CK. Z czasem zostało jednak rozszerzone i jego aktualna wersja wylicza wszystkie badane tu metryki produktu.

W celu zgromadzenia danych do opisanych w kolejnych rozdziałach eksperymentów opracowano nową wersję narzędzia *Ckjm*. Podstawę stanowiła wersja 1.8*. Wersja ta potrafiła wyliczyć sześć metryk z zestawu CK oraz metryki Ce i NPM. Na podstawie tej wersji opracowano rozszerzoną wersję programu†, która wylicza kolejnych jedenaście metryk wymienionych w Rozdziale 3.2, co daje w sumie dziewiętnaście różnych metryk oprogramowania. Mimo stosunkowo dużej liczby metryk program zachował swoje początkowe zalety, czyli prostotę i szybkość działania. Program integruje się z *Apache Ant* co umożliwia łatwe jego wdrożenie w środowiskach programistycznych opartych zarówno na *Apache Ant* jak i *Apache Maven*. Działanie programu zostało szczegółowo przeanalizowane i zweryfikowane pod kątem możliwości zastosowania w modelach predykcji defektów [55].

3.4.2 BugInfo

W celu zgromadzenia danych dotyczących metryk procesu i liczby defektów opracowano narzędzie *BugInfo*‡. Narzędzie to analizuje zawartość systemu kontroli wersji (aktualna wersja *BugInfo* współpracuje z *CVS* oraz *SubVersion*) ze szczególnym uwzględnieniem komentarzy wstawianych podczas zatwierdzania modyfikacji. Interpretacja komentarzy była konieczna do wykrywania defektów, metryki produktu są albo niemal bezpośrednio dostępne w systemie kontroli wersji albo opierają się na defektach, jak na przykład NDPV. Jeżeli komentarz wskazuje, że zatwierdzane zmiany dotyczą naprawiania defektu, przyjmuje się, że modyfikowane klasy zawierały defekt przed zatwierdzeniem tych zmian.

Każdy z badanych projektów został przeanalizowany pod kątem reguł, według których komentowano modyfikacje powiązane z naprawianiem defektów. W niemal wszystkich przypadkach tego typu modyfikacje było komentowane przy pomocy identyfikatora (lub adresu url) defektu w systemie śledzenia defektów albo przy pomocy pewnych słów kluczowych, takich jak na przykład *bugfix*. Na podstawie tych analiz, dla każdego z projektów skonstruowano wyrażenie regularne, które następnie zostało przekazane jako parametr do programu *BugInfo*. Jeżeli komentarz towarzyszący modyfikacji pasował do przekazanego wyrażenia regularnego, *BugInfo* identyfikował modyfikację jako zmianę naprawiającą defekt i tym samym zwiększał licznik defektów dla modyfikowanej klasy. Tego typu podejście jest powszechnie uznawane i rekomendowane w pracach dotyczących predykcji defektów, na przykład w [42].

Poprawność działania narzędzia została zweryfikowana przy pomocy wyczerpujących testów funkcjonalnych i jednostkowych. Testy te zostały zautomatyzowane do postaci testów JUnit i zostały opublikowane w internecie wraz z wersją wykonywalną programu. *BugInfo* został już wykorzystany w szeregu prac naukowych: [46], [47], [48], [49] oraz [55].

3.4.3 Metrics Repository

Na potrzeby opisanych w kolejnych rozdziałach eksperymentów zostały zebrane dane z 92 wersji 38 projektów. Jest to ilość niebagatelna jeżeli zwrócić uwagę na fakt, że w największym ogólnodostępnym repozytorium metryk – *Promise*§ zestawy metryk opisują od 1 do 9 wersji projektów, a wszystkie

*http://www.spinellis.gr/sw/ckjm/
†http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/
‡http://kenai.com/projects/buginfo
§http://promisedata.org/

zestawy łącznie zawierają informacje o 57 wersjach projektów (stan po konferencji PROMISE'09). W repozytorium Promise w poszczególnych zestawach wyliczano różne metryki, więc w ramach jednego eksperymentu trzeba się zazwyczaj ograniczać do jednego zestawu, co daje maksymalnie 9 wersji projektów.

Duża część z zebranych na potrzeby tej pracy metryk została wykorzystana w pracy przyjętej na konferencję PROMISE'10 [47] i w związku z tym zostanie w niedalekiej przyszłości dołączona do repozytorium *Promise*. Natomiast już teraz działa witryna internetowa[¶], na której zebrano metryki dotyczące wszystkich wersji wszystkich projektów analizowanych w ramach któregokolwiek z eksperymentów opisanych w kolejnych rozdziałach.

[¶]http://purl.org/MarianJureczko/MetricsRepo

Rozdział 4

Czynniki wpływające na dokładność predykcji defektów

W rozdziale tym przedstawiono wyniki eksperymentów badających użyteczność poszczególnych metryk w modelach predykcji defektów. Badano tu wszystkie wymienione w Rozdziale 2.2 metryki procesu oraz dwie metryki produktu reprezentujące rozmiar: *WMC* oraz *LOC*. Badanie metryki polegało na dodaniu jej do zestawu dwudziestu, wymienionych w Rozdziale 2.2 metryk produktu i skonstruowaniu na podstawie uzyskanego w taki sposób zestawu metryk modelu. Model ten następnie porównywano z modelem skonstruowanym bez użycia badanej metryki. Jeżeli model wykorzystujący badaną metrykę okazywał się być lepszy, to uznawano badaną metrykę za użyteczną w predykcji defektów. Analizowano dwie różne metody wprowadzania badanej metryki do modelu, a model konstruowano przy pomocy postępującej regresji liniowej.

Duża część z przedstawionych tu wyników została już opracowana i wcześniej zaprezentowana w [49] oraz [55].

4.1 Definicja eksperymentu

Przeprowadzano dwa typy eksperymentów badających użyteczność poszczególnych metryk. W obu typach eksperymentów wprowadzano badaną metrykę do modelu predykcji defektów, a następnie porównywano uzyskany model z modelem pozbawionym badanej metryki. W dalszej części tego rozdziału modele wykorzystujące badaną metrykę będą nazywane modelami złożonymi, a modele jej nie wykorzystujące modelami prostymi. Różnica pomiędzy wspomnianymi dwoma typami eksperymentów wynikała z metody wprowadzania badanej metryki do modelu. Wyróżnić można metodę I oraz metodę II.

Załóżmy, że w_i jest wydaniem numer i danego projektu. Niech Mw_i będzie prostym modelem predykcji defektów, zbudowanym na podstawie danych pochodzących z wydania w_i bez wykorzystywania badanej metryki. Niech $M'w_i$ będzie złożonym modelem predykcji defektów, zbudowanym na podstawie danych pochodzących z wydania w_i, do którego wprowadzano badaną metrykę wykorzystując metodę I. Niech $M''w_i$ będzie złożonym modelem predykcji defektów, zbudowanym na podstawie danych pochodzących z wydania w_i, do którego wprowadzano badaną metrykę wykorzystując metodę II.

W celu zbudowania prostego modelu predykcji defektów stosowano wszystkie dwadzieścia metryk produktu (szczegóły na ich temat znajdują się w Rozdziale 2.2). Następnie stosowano postępującą regresję liniową, aby wyrazić zależność pomiędzy tymi metrykami a liczbą historycznych defektów.

Dzięki temu, że stosowano regresję postępującą uzyskiwane modele nie zawierały wszystkich dwudziestu metryk, a jedynie te najistotniejsze. Typowy, prosty model predykcji defektów wykorzystywał od pięciu do dziesięciu metryk. Zarówno proste, jak i złożone modele dokonują predykcji liczby defektów dla każdej z klas wchodzących w skład danego wydania projektu z osobna.

W celu zbudowania modelu złożonego M', czyli takiego, do którego wprowadzano badaną metrykę metodą I, zestaw dwudziestu metryk produktu rozszerzano o badaną metrykę. Następnie stosowano tą samą procedurę co w przypadku modelu prostego, czyli postępującą regresję liniową.

Aby zbudować model złożony M'', czyli taki, do którego wprowadzano badaną metrykę metodą II, trzeba było najpierw zdefiniować nową, pomocniczą metrykę – nm. Niech bm będzie badaną metrykę, która ma zostać wprowadzona do modelu metodą II. Niech nm będzie metryką binarną taką, że będzie równa 0 dla tych klas, dla których metryka pm będzie mniejsza od mediany metryki pm w rozważanym wydaniu projektu. W pozostałych przypadkach metryka nm będzie równa 1. Metryka nm musi zostać wyliczona dla każdej klasy rozważanego wydania projektu. Niech MP będzie zbiorem dwudziestu metryk produktu, które wcześniej były wykorzystywane do konstrukcji modelu prostego. Niech MP' będzie zbiorem metryk skonstruowanych przy pomocy metryki nm, w taki sposób, że dla każdej metryki $mp \in MP$ utworzono metrykę $mp' \in MP'$, taką że $mp' = mp * nm$. Następnie konstruowano model predykcji defektów M'' wykorzystując dwadzieścia metryk produktu należących do zbioru MP oraz dwadzieścia nowo utworzonych metryk należących do zbioru MP'. Do konstrukcji modelu używano postępującej regresji liniowej.

Motywacją stojącą za metodą II wprowadzania metryk procesu do modelu była chęć zweryfikowania istnienia nieciągłości w związkach pomiędzy metrykami produktu a liczbą defektów. Nieciągłości, które mogłyby zostać wyjaśnione przy pomocy badanych metryk. Na przykład, jeżeli jest prawdą, że w klasach modyfikowanych przez wielu różnych autorów (metryka NDC) liczbę defektów można najlepiej przewidywać przy pomocy metryki RFC, a w klasach, które były modyfikowane przez co najwyżej jedną osobę, przy pomocy metryki LCOM, to najlepszych predykcji dostarczy właśnie model M'', do którego wprowadzono metrykę NDC metodą II.

Przyjmujemy, że $E(Mw_i, w_{i+1})$ jest wynikiem ewaluacji efektywności modelu Mw_i w dokonywaniu predykcji w wydaniu w_{i+1}; $E(M'w_i, w_{i+1})$ wynikiem ewaluacji efektywności modelu $M'w_i$ w dokonywaniu predykcji w wydaniu w_{i+1}; $E(M''w_i, w_{i+1})$ wynikiem ewaluacji efektywności modelu $M''w_i$ w dokonywaniu predykcji w wydaniu w_{i+1}. Warto tu podkreślić, że zawsze używano modelu skonstruowanego na danych chronologicznie starszych do wykonania predykcji na danych nowszych, pochodzących z tego samego projektu.

Niech c_1, c_2, ..., c_n reprezentują klasy z wydania w ułożone w porządku malejącym według predykcji uzyskanych z modelu M, n to liczba klas. Niech d_1, d_2, ..., d_n będą liczbami rzeczywistych defektów występujących w kolejnych klasach (porządek ułożenia klas nie ulega zmianie). Niech D_j będzie liczbą defektów w j początkowych klasach: $D_j = \sum_{i=1}^{j} d_i$. Niech k będzie najmniejszym indeksem takim, że $D_k > 0,8 * D_n$, wtedy ocenę modelu będzie można wyliczyć z następującego wzoru:

$$E(M, w) = \frac{k}{n} \times 100\% \tag{4.1}$$

Aby ocenić, czy badane metryki są rzeczywiście przydatne w predykcji defektów, czyli czy zbudowane z ich pomocą modele dały wyniki istotnie lepsze, sformułowano i zastosowano w stosunku do każdej z badanych metryk z osobna następujące hipotezy:

- $H_{0,M'}$ – Nie ma istotnej różnicy pomiędzy dokładnościami predykcji uzyskanymi z modeli prostych M a dokładnościami predykcji uzyskanymi z modeli złożonych M', zbudowanych z wykorzystaniem badanej metryki.

- $H_{0,M''}$ – Nie ma istotnej różnicy pomiędzy dokładnościami predykcji uzyskanymi z modeli prostych M a dokładnościami predykcji uzyskanymi z modeli złożonych M'', zbudowanych z wykorzystaniem badanej metryki.

Sformułowano następujące hipotezy alternatywne:

- $H_{1,M'}$ – Istnieje istotna różnica pomiędzy dokładnościami predykcji uzyskanymi z modeli prostych M a dokładnościami predykcji uzyskanymi z modeli złożonych M', zbudowanych z wykorzystaniem badanej metryki.

- $H_{1,M''}$ – Istnieje istotna różnica pomiędzy dokładnościami predykcji uzyskanymi z modeli prostych M a dokładnościami predykcji uzyskanymi z modeli złożonych M'', zbudowanych z wykorzystaniem badanej metryki.

W celu zweryfikowania hipotez stosowano test t dla prób zależnych. Stosowanie tego testu wymaga spełnienia kilku założeń. Test ten mianowicie wymaga, aby badane wielkości były mierzone przynajmniej na skali interwałowej, aby dokonywane obserwacje były niezależne i aby badane próbki miały homogeniczną wariancję oraz aby przyjmowane przez nie wartości miały rozkład normalny. Dwa pierwsze założenia były spełnione we wszystkich przypadkach. Założenie o homogeniczności wariancji było sprawdzane przy pomocy testu Levenea, a normalność rozkładu przy pomocy testu Shapiro–Wilka. Zarówno homogeniczność wariancji jak i normalność były sprawdzane na poziomie istotności $\alpha = 0,05$. Jeżeli któreś z założeń nie byłoby spełnione zastosowanoby test Wilcoxona dla par obserwacji. Hipotezy były testowane na poziomie istotności $\alpha = 0,05$.

W eksperymentach, w których uzyskane modele złożone dawały zdecydowanie lepsze predykcje niż modele proste dodatkowo wyliczono siłę efektu. Do tego celu posłużono się procedurą przedstawioną w [72]:

$$d = t \times \sqrt{\frac{2 \times (1 - r)}{n}} \tag{4.2}$$

gdzie r to współczynnik korelacji Pearsona, t to wartość statystyki t, a n to rozmiar próbki. Następnie wartość siły efektu uzyskano ze wzoru zaproponowanego przez Cohena [14]:

$$r_{pb} = \frac{d}{\sqrt{d^2 + 4}} \tag{4.3}$$

Uzyskana w ten sposób wartość była porównywana ze wzorcem zaproponowanym przez Kampenesa [56], który definiuje wartości nominalne opisujące siłę efektu: słaby ($0 \leqslant r_{pb} < 0,193$), średni($0,193 \leqslant r_{pb} < 0,456$), mocny($0,456 \leqslant r_{pb} \leqslant 0,868$).

4.2 Wyniki eksperymentu

Wyniki eksperymentu zostały przedstawione dla każdego z badanych czynników z osobna. Warto tu przypomnieć, że w zastosowanej metodzie oceniania modeli mniejsza wartość oceny $E(M)$ oznacza lepszy model. Zaprezentowano uzyskane oceny efektywności modeli oraz wyniki testów hipotez statystycznych.

4.2.1 Wyniki dla metryki NR

Metrykę NR badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów.

Metryka NR wprowadzana do modelu metodą I. Dwadzieścia cztery wydania projektów zostały przeanalizowane w celu ocenienia użyteczności metryki NR. Zgodnie z tabelą 4.1 średnio 59,43% klas musi zostać przetestowanych w celu wykrycia 80% defektów, gdy stosowany jest model prosty. Natomiast używając modelu złożonego M' liczbę klas, które trzeba przetestować aby osiągnąć identyczne wyniki w wykrywaniu defektów udało się zredukować do $54,39\%$. Okazuje się jednak, że różnica taka nie jest istotna statystycznie przy $\alpha = 0,05$. Jak można zobaczyć w Tabeli 4.2 uzyskane prawdopodobieństwo testowe wyniosło $0,075$. W związku z tym nie ma podstaw do odrzucenia hipotezy $H_{0,M'}$, co oznacza, że użyteczność metryki NR w modelach predykcji defektów jest ograniczona. Aczkolwiek, z uwagi na fakt, że średnia ocena modeli wykorzystujących metrykę NR była o 5% mniejsza (w przypadku mediany różnica ta wyniosła aż 9%), warto tą metrykę brać po uwagę.

Z uwagi na stosunkowo dużą różnicę pomiędzy wynikami dla modeli prostych, a złożonych wyliczono siłę efektu. Otrzymano $r_{pb} = 0,11$, co według Kampenesa [56] oznacza słaby efekt.

Metryka NR wprowadzana do modelu metodą II. Wprowadzanie metryki NR metodą II zaowocowało modelami złożonymi, które dawały predykcje minimalnie dokładniejsze niż modele proste, co pokazano w Tabeli 4.3. Różnica w dokładności była na tyle niewielka, że nie była istotna statystycznie (Tabela 4.4). Nie można więc stwierdzić, że wstawianie metryki NR do modelu predykcji defektów metodą II jest działaniem poprawiającym jakość predykcji.

4.2.2 Wyniki dla metryki NDC

Metrykę *NDC* badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów.

Metryka NDC wprowadzana do modelu metodą I. Zastosowanie metryki *NDC* dało bardzo ciekawe wyniki. Dzięki wprowadzeniu tej metryki metodą I uzyskane zostały modele dające zdecydowanie lepsze predykcje. Modele wykorzystujące metrykę *NDC* pozwalały na zidentyfikowanie tej samej liczby defektów co modele proste w zdecydowane mniejszej liczbie klas. Tabela 4.5 pokazuje tę różnicę zarówno w przypadku średniej oceny, jak i mediany. Co więcej, okazało się, że różnica ta była istotna statystycznie – Tabela 4.6. Udało się zatem dla metryki *NDC* odrzucić hipotezę zerową $H_{0,M'}$ i przyjąć alternatywną $H_{1,M'}$. Tym samym można stwierdzić, że istnieje istotna różnica pomiędzy dokładnościami predykcji uzyskanymi z modeli prostych M a dokładnościami predykcji uzyskanymi z modeli złożonych M' wykorzystujących metrykę *NDC* wprowadzoną metodą I.

Dla modeli wykorzystujących metrykę *NDC* wprowadzoną metodą I wyliczono siłę efektu $r_{pb} = 0,20$. Wartość taka według Kampenesa [56] wskazuje na średnią siłę efektu, co dodatkowo potwierdza istotność metryki NDC w predykcji defektów.

Metryka NDC wprowadzana do modelu metodą II. Wprowadzanie metryki *NDC* do modelu predykcji defektów metodą II dało nieco gorsze rezultaty niż metodą I. W tym przypadku modele złożone były jedynie minimalnie lepsze od modeli prostych. Zgodnie z Tabelą 4.7 średnia ocena modelu prostego wynosi $56,91$, a modelu zaawansowanego $54,77$. Różnica taka okazała się nie być istotną statystycznie.

Tabela 4.1: Number of Revisions, metoda I – oceny modeli

	Średnia	Mediana	σ
$E(M)$	59,43	61,03	16,73
$E(M')$	54,39	52,04	23,03

Tabela 4.2: Number of Revisions, metoda I – testy hipotez

		$E(M)$	$E(M')$
Test	W	0,99	0,98
Shapiro–Wilka	p	0,94	0,17
Test	df	46	
Levenea	F(1,df)	3,92	
	p	0,054	
	t	1,87	
Test–t	df	23	
	p	**0,075**	

Tabela 4.3: Number of Revisions, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	56,73	58,72	17,48
$E(M'')$	54,39	58,53	20,23

Tabela 4.4: Number of Revisions, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,97	0,95
Shapiro–Wilka	p	0,51	0,20
Test	df	60	
Levenea	F(1,df)	1,18	
	p	0,28	
	t	1,37	
Test–t	df	30	
	p	**0,182**	

Tabela 4.5: Number of Distinct Commiters, metoda I – oceny modeli

	Średnia	Mediana	σ
$E(M)$	58,51	56,98	12,84
$E(M')$	51,01	51,01	18,60

Tabela 4.6: Number of Distinct Commiters, metoda I – testy hipotez

		$E(M)$	$E(M')$
Test	W	0,98	0,95
Shapiro–Wilka	p	0,96	0,43
Test	df	34	
Levenea	F(1,df)	2,93	
	p	0,10	
	t	3,02	
Test–t	df	17	
	p	**0,007**	

Jak pokazuje Tabela 4.8 uzyskano prawdopodobieństwo testowe $p = 0,192$, co nie pozwoliło na od-

Tabela 4.7: Number of Distinct Commiters, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	56,91	59,37	17,22
$E(M'')$	54,77	57,84	20,26

Tabela 4.8: Number of Distinct Commiters, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,97	0,44
Shapiro–Wilka	p	0,95	0,10
Test	df	62	
Levenea	F(1,df)	1,53	
	p	0,22	
	t	1,33	
Test–t	df	31	
	p	**0,192**	

rzucenie hipotezy zerowej. Nie ma więc podstaw do zalecania stosowania metody II do wprowadzania metryki *NDC* do modelu predykcji defektów.

4.2.3 Wyniki dla metryki NML

Metrykę *NML* badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów.

Tabela 4.9: Number of Modified Lines, metoda I – oceny modeli

	Średnia	Mediana	σ
$E(M)$	55,06	56,45	12,47
$E(M')$	53,36	55,42	13,38

Tabela 4.10: Number of Modified Lines – testy hipotez

		$E(M)$	$E(M')$
Test	W	0,97	0,94
Shapiro–Wilka	p	0,88	0,48
Test	df	26	
Levenea	F(1,df)	0,28	
	p	0,60	
	t	2,74	
Test–t	df	13	
	p	**0,017**	

Metryka NML wprowadzana do modelu metodą I. Na podstawie Tabeli 4.9 można by stwierdzić, że wprowadzenie metryki *NML* do modelu predykcji defektów metodą I tylko nieznacznie poprawiło dokładność predykcji. Jednak mimo niezbyt licznej próby (predykcji próbowano dokonywać jedynie w 14 wydaniach projektów) uzyskano wynik istotny statystycznie na poziomie istotności $\alpha = 0,05$.

Jak można zobaczyć w Tabeli 4.10 uzyskano prawdopodobieństwo testowe $p = 0,017$, co pozwoliło na odrzucenie hipotezy zerowej i zaakceptowanie alternatywnej. Uzyskane wyniki pozwalają więc na stwierdzenie, że metryka *NML* jest użyteczna w predykcji defektów, jeżeli wprowadzać ją do modelu metodą I.

Z uwagi na fakt, że uzyskano wynik istotny statystycznie wyliczono dodatkowo siłę efektu. Uzyskano wartość $r_{pb} = 0,06$ co według Kampenesa [56] odpowiada słabemu efektowi.

Tabela 4.11: Number of Modified Lines, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	56,71	58,31	11,26
$E(M'')$	55,09	56,93	13,76

Tabela 4.12: Number of Modified Lines, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,98	0,98
Shapiro–Wilka	p	0,99	0,97
Test	df	24	
Levenea	F(1,df)	0,44	
	p	0,51	
	t	0,92	
Test–t	df	12	
	p	**0,376**	

Metryka NML wprowadzana do modelu metodą II. Zgodnie z Tabelą 4.11 wprowadzenie do modelu predykcji defektów metryki *NML* metodą II dało pozytywny efekt. Dokładność modeli zwiększyła się, jednak jak wynika z Tabeli 4.12 poprawa nie była istotna statystycznie. Uzyskano stosunkowo wysokie prawdopodobieństwo testowe $p = 0,376$, co nie pozwoliło na odrzucenie hipotezy zerowej. Tym samym brak przekonywujących argumentów do tego aby postulować wprowadzanie metryki *NML* metodą II do modeli predykcji defektów.

4.2.4 Wyniki dla metryki IN

Metryka *IN* jest metryką binarną. W związku z tym do jej badania zastosowano tylko metodę II.

Tabela 4.13: Is New, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	56,00	58,52	17,67
$E(M')$	61,00	60,94	16,52

Metryka IN wprowadzana do modelu metodą II. Tabela 4.13 pokazuje, że wprowadzenie metryki *IN* metodą II doprowadziło do powstania modeli dających gorsze predykcje niż modele proste. Dodatkowo, na podstawie Tabeli 4.14 można stwierdzić, że pogorszenie predykcji było istotne statystycznie. Konkluzją jest więc stwierdzenie negatywnego wpływu metryki *IN* na modele predykcji defektów i sformułowanie rekomendacji odradzającej stosowanie tej metryki.

Tabela 4.14: Is New – testy hipotez

		$E(M)$	$E(M')$
Test	W	0,97	0,98
Shapiro–Wilka	p	0,47	0,80
Test	df	62	
Levenea	F(1,df)	0,24	
	p	0,63	
	t	-3,01	
Test–t	df	31	
	p	**0,005**	

4.2.5 Wyniki dla metryki NDPV

Metrykę *NDPV* badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów.

Tabela 4.15: Number of Defects in Previous Version, metoda I – oceny modeli

	Średnia	Mediana	σ
$E(M)$	55,77	58,27	12,47
$E(M')$	55,87	55,42	13,38

Tabela 4.16: Number of Defects in Previous Version, metoda I – testy hipotez

		$E(M)$	$E(M')$
Test	W	0,97	0,95
Shapiro–Wilka	p	0,55	0,14
Test	df	64	
Levenea	F(1,df)	0,80	
	p	0,37	
	t	-0,06	
Test–t	df	32	
	p	**0,953**	

Metryka NDPV wprowadzana do modelu metodą I. Wprowadzenie metryki *NDPV* do modeli predykcji defektów przy pomocy metody I nie wpłynęło w znaczący sposób na uzyskiwane predykcje. Jak pokazuje Tabela 4.15 średnia ocena modeli prostych i modeli zaawansowanych wykorzystujących metrykę *NDPV* jest niemal identyczna. Nie powinno więc również dziwić, że jak można zauważyć w Tabeli 4.16, różnica ta nie jest istotna statystycznie.

Tabela 4.17: Number of Defects in Previous Version, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	55,66	58,27	16,90
$E(M'')$	54,71	57,48	17,46

Tabela 4.18: Number of Defects in Previous Version, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,97	0,97
Shapiro–Wilka	p	0,59	0,61
Test	df	60	
Levenea	F(1,df)	0,21	
	p	0,65	
	t	0,62	
Test–t	df	30	
	p	**0,541**	

Metryka NDPV wprowadzana do modelu metodą II. Wprowadzenie metryki *NDPV* metodą II dało, jak pokazuje Tabela 4.17, odrobinę lepsze wyniki niż w przypadku metody I. Poprawa jednak nie była na tyle duża, żeby doprowadziła do uzyskania wyników istotnych statystycznie – Tabela 4.18. Prawdopodobieństwo testowe p wyniosło $0,541$ w związku z czym nie było podstaw do odrzucenia hipotezy zerowej. Uzyskane wyniki nie dają więc podstaw do rekomendowania stosowania metryki *NDPV* bez względu na to jaką metodą metryka ta by nie była wprowadzona do modelu predykcji defektów.

4.2.6 Wyniki dla metryki NER

Metrykę *NER* badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów. Z uwagi na problemy natury technicznej, związane ze zbieraniem tej metryki, badania jej dotyczące obejmują tylko jeden projekt. Jest to projekt *Przemysłowy-2*.

Tabela 4.19: Number of Evening Revisions, metoda I – oceny modeli

	Średnia	Mediana	σ
$E(M)$	50,88	50,74	12,62
$E(M')$	48,18	40,49	16,40

Metryka NER wprowadzana do modelu metodą I. Metrykę *NER* badano w zbyt małej liczbie wydań projektów, aby można było zweryfikować statystyczną istotność uzyskanych wyników. Natomiast uzyskane i przedstawione w Tabeli 4.19 wyniki ewaluacji modeli pokazują, że w przebadanych wydaniach projektów metryka ta pozytywnie wpłynęła na dokładność predykcji. Zarówno średnia jak i mediana z ocen modeli jest wyraźnie lepsza w przypadku modeli złożonych. Wyniki takie zachęcają do podjęcia dalszych badań nad tą metryką, które pozwoliłyby w jednoznaczny sposób ocenić jej wpływ.

Tabela 4.20: Number of Evening Revisions, metoda II - oceny modeli

	Średnia	Mediana	σ
$E(M)$	50,88	50,74	12,62
$E(M'')$	51,34	51,35	10,37

Metryka NER wprowadzana do modelu metodą II. Wprowadzenie metryki *NER* metodą II doprowadziło do pogorszenia dokładności predykcji. Wydaje się więc, że użyteczność tej metryki ogranicza się

do wprowadzania jej metodą I. Zdecydowanie nie są to jednak wyniki rozstrzygające z uwagi na zbyt małą liczbę przebadanych wydań projektów.

4.2.7 Wyniki dla metryki NPR

Metrykę *NPR* badano korzystając z obu opisanych wcześniej metod wprowadzania metryki do modelu predykcji defektów. Z uwagi na problemy natury technicznej związane ze zbieraniem tej metryki, badania jej dotyczące obejmują tylko trzy wydania projektu *Przemysłowy-2*. Ocenę modeli można natomiast było przeprowadzić tylko dla dwóch wydań.

Tabela 4.21: Number of Pre–code–freeze Revisions – oceny modeli

Numer wydania	$E(M)$	$E(M')$	$E(M'')$
3	36,05	36,05	46,82
4	54,61	56,88	56,19

Metryka *NPR* doprowadziła do pogorszenia dokładności predykcji modeli zarówno wtedy, kiedy wprowadzano ją metodą I, jak i wtedy kiedy wprowadzano ją metodą II. Uzyskane wyniki nie są rozstrzygające z uwagi na niewielką liczbę przebadanych wydań projektów, ale wskazują, że użyteczność tej metryki w predykcji defektów jest dyskusyjna.

4.2.8 Wyniki dla metryki WMC

Metryka *WMC* jest pierwszą z dwóch metryk reprezentujących rozmiar klasy, których użyteczność badano. Metryka ta jest jedną z metryk produktu, które wchodzą w skład modelu prostego. W związku z tym nie można jej wprowadzać metodą I, ponieważ jest ona już obecna w zestawie metryk wykorzystywanym do budowy modelu. Dlatego przy jej analizowaniu wykorzystano tylko metodę II.

Tabela 4.22: Weighted Methods per Class, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	55,37	57,41	18,66
$E(M'')$	55,97	59,21	17,80

Tabela 4.23: Weighted Methods per Class, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,98	0,66
Shapiro–Wilka	p	0,98	0,46
Test	df	98	
Levenea	F(1,df)	0,08	
	p	0,77	
	t	-0,81	
Test–t	df	49	
	p	**0,422**	

Metryka WMC wprowadzana do modelu metodą II. Wprowadzenie metryki *WMC* do modeli predykcji defektów metodą II doprowadziło do nieznacznego pogorszenia dokładności predykcji. Jak pokazuje

Tabela 4.22 średnia ocena modeli prostych jest niemal identyczna ze średnią oceną modeli złożonych. Nie może więc dziwić, że różnica nie była istotna statystycznie. Zgodnie z Tabelą 4.23 prawdopodobieństwo testowe wyniosło $0,422$. Nie było więc żadnych podstaw do odrzucenia hipotezy zerowej.

4.2.9 Wyniki dla metryki LOC

Metryka *LOC* jest drugą z dwóch metryk reprezentujących rozmiar klasy, których użyteczność badano. Metryka ta jest jedną z metryk produktu, które wchodzą w skład modelu prostego. W związku z tym nie można jej wprowadzać metodą I, ponieważ jest ona już obecna w zestawie metryk wykorzystywanym do budowy modelu. Dlatego przy jej analizowaniu wykorzystano tylko metodę II.

Tabela 4.24: Lines of Code, metoda II – oceny modeli

	Średnia	Mediana	σ
$E(M)$	55,17	56,54	18,50
$E(M'')$	53,19	52,04	17,20

Tabela 4.25: Lines of Code, metoda II – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,98	0,98
Shapiro–Wilka	p	0,75	0,68
Test	df	100	
Levenea	F(1,df)	0,29	
	p	0,59	
	t	2,24	
Test–t	df	50	
	p	**0,029**	

Metryka LOC wprowadzana do modelu metodą II. Metryka *LOC* jest drugą obok metryki *WMC* metryką reprezentującą rozmiar. Jak pokazano w Rozdziale 2.3 istnieją silne argumenty na rzecz tezy mówiącej, iż metryki reprezentujące rozmiar mogą w dużym stopniu wyjaśniać niedokładności modeli skonstruowanych na podstawie metryk produktu. Nie potwierdził tej tezy eksperyment dotyczący metryki *WMC*. Inaczej jest jednak w przypadku metryki *LOC*. Jak można zobaczyć w Tabeli 4.24, wprowadzenie metryki *LOC* do modeli predykcji defektów metodą II poprawiło dokładność predykcji. Dodatkowo Tabela 4.25 pokazuje, że poprawa ta była istotna statystycznie. Można więc odrzucić hipotezę zerową, przyjąć hipotezę alternatywną i zarekomendować wprowadzanie metryki *LOC* do modeli predykcji defektów przy pomocy metody II.

Uzyskano wyniki istotne statystycznie, więc wyliczono również siłę efektu jaki spowodowało wprowadzenie metryki *LOC*. Wyliczona siła efektu wyniosła: $r_{pb} = 0,07$, co według Kampenesa [56] odpowiada słabemu efektowi.

4.2.10 Wyniki dla kombinacji czterech metryk procesu

Zestaw czterech metryk procesu (NR, NDC, NML, NDPV) badano wprowadzając je do modelu predykcji defektów metodą I. Zastosowanie metody II do równoczesnego wprowadzenia aż czterech metryk spowodowałby eksplozję liczby wszystkich metryk wykorzystywanych podczas budowy modelu.

Tabela 4.26: Kombinacja metryk procesu – oceny modeli

	Średnia	Mediana	σ
$E(M)$	56,53	58,27	11,67
$E(M')$	52,43	59,31	15,99

Tabela 4.27: Kombinacja metryk procesu – testy hipotez

		$E(M)$	$E(M'')$
Test	W	0,98	0,89
Shapiro–Wilka	p	0,98	0,11
Test	df	24	
Levenea	F(1,df)	1,48	
	p	0,24	
	t	1,71	
Test–t	df	12	
	p	**0,113**	

Kombinacja metryk wprowadzana do modelu metodą I. Bardzo ciekawe efekty uzyskano w przypadku modelu wykorzystującego kombinację czterech metryk procesu. W przypadku trzech, spośród tych czterech metryk, wprowadzanie ich pojedynczo prowadziło do zauważalnej poprawy dokładności predykcji. W dwóch przypadkach była ona istotna statystycznie. Okazuje się jednak, że zastosowanie kombinacji tych metryk nie doprowadziło do dalszej poprawy, a wręcz przeciwnie. Jak można zauważyć w Tabeli 4.26 średnia ocena modelu jest lepsza niż średnia ocena modelu prostego, ale w przypadku mediany relacja ta jest odwrotna. Nie jest zatem sensacją, że wynik taki nie okazał się być istotny statystycznie. Zaprezentowane w Tabeli 4.27 prawdopodobieństwo testowe p zdecydowanie przekracza przyjęty poziom istotności $\alpha = 0,05$. Nie ma zatem podstaw do odrzucenia hipotezy zerowej.

Dla modeli wykorzystujących kombinację metryk procesu wyliczono siłę efektu. Jak w świetle wyników przedstawionych w Tabelach 4.26 i 4.27 można było oczekiwać, uzyskany efekt był słaby. Wartość r_{pb} wyniosła $0,13$.

4.3 Zagrożenia dla ważności uzyskanych wyników

Wohlin i in. [117] wyróżnili cztery rodzaje walidacji w analizie zagrożeń dla ważności uzyskanych wyników. Są to walidacja wewnętrzna (ang. internal validity), walidacja zewnętrzna (ang. external validity), walidacja konkluzji (ang. conclusion validity) oraz walidacja konstrukcji (ang. construct validity).

Walidacja wewnętrzna jest skoncentrowana na ważności danego eksperymentu. Analizuje istnienie związku przyczynowo – skutkowego pomiędzy zmiennymi niezależnymi a zależnymi. Walidację wewnętrzną można uznać za nienaruszoną jeżeli następujące warunki są spełnione:

- przyczyna poprzedza skutek,
- istnieje statystyczna zależność pomiędzy przyczyną a skutkiem,
- brakuje alternatywnych, wiarygodnych wyjaśnień zaobserwowanych zjawisk.

Najistotniejszym zagrożeniem dla walidacji wewnętrznej jest występowanie zmiennych wpływających na zmienną zależną, których istnienia badacz nie jest świadomy i w związku z tym nie uwzględnił ich w eksperymencie.

Można stwierdzić, że uzyskane wyniki posiadają walidację zewnętrzną wtedy, kiedy można je odnieść nie tylko do badanej próbki, ale również do całej populacji. Problemy z walidacją zewnętrzną pojawiają się zazwyczaj przy małych próbkach, w szczególności takich, które składają się z elementów mających jakąś wspólną cechę. Nie sposób w takich sytuacjach rozstrzygnąć, czy zaobserwowane wyniki mają zastosowanie do przedstawicieli populacji nie posiadających tej cechy.

Walidacja konkluzji to poziom, do którego empirycznie odkryte w analizowanej próbce zależności są rozsądne i wiarygodne. Konkluzja może być obarczona jednym z dwóch rodzajów błędów. Można uznać, że nie istnieje relacja pomiędzy badanymi zjawiskami wtedy, kiedy w rzeczywistości taka relacja występuje. Albo można doszukać się zależności tam, gdzie jej w rzeczywistości nie ma. Podstawowymi zagrożeniami walidacji związanymi z błędami pierwszego rodzaju są szumy. Według Trochima i Donnellego [106] szumy mogą być wynikiem: mało wiarygodnych pomiarów, nietrzymania się procedury eksperymentu, losowego występowania czynników niezwiązanych z badaniem czy też dużej różnorodności w badanej próbce. Błędy drugiego rodzaju są efektem "złowienia" zależności, która przypadkowo zaszła w badanej próbce. W celu uniknięcia tego zagrożenia należy zadbać o niezależność poszczególnych eksperymentów albo stosować korektę Bonferroniego [46]. Źródłem obu rodzajów błędów może być zaniechanie sprawdzania założeń dotyczących badanych danych.

Walidacja konstrukcji dotyczy stopnia, do którego wyciągnięte z badania empirycznego wnioski mogą być zastosowane do teorii, na której to badanie empiryczne się opierało [106]. Jest więc, podobnie jak walidacja zewnętrzna, związana z generalizacją wyników. Tu jednak generalizację ograniczamy do teorii lub konceptu, na którym opierało się badanie. Według Trochima i Donnellego [106] najistotniejsze zagrożenia walidacji konstrukcji to: niewystarczające objaśnienie przedmiotu badania, zbytnie zawężenie przedmiotu badań lub metody badawczej, interakcje, zbytnie generalizowanie wyników czy też zgadywanie hipotez na podstawie danych.

4.3.1 Walidacja wewnętrzna

Wymieniono powyżej trzy warunki, które należy spełnić, aby walidacja wewnętrzna nie była naruszona. W omawianym w tym rozdziale eksperymencie żaden z tych trzech warunków nie został złamany. Jak podają D'Ambros i in. [16], zagrożenia dla wewnętrznej walidacji badań empirycznych w inżynierii oprogramowania wynikają w głównej mierze z ludzkich interakcji. W rozważanym tu eksperymencie czynniki takie nie występowały. Najpopularniejszym czynnikiem ludzkim w badaniach dotyczących modeli predykcji defektów jest istotność (srogość) defektu. Jest to atrybut, który jest wprowadzany do systemu śledzenia defektów przez testera lub programistę i zachodzi zagrożenie, że jego wartość może zostać zmanipulowana przez osobę pracującą nad defektem w celu uzyskania osobistych korzyści. Dodatkowo warto zauważyć, że metryki istotności defektów zostały skrytykowane przez Ostranda i in. [88], jako bardzo subiektywne i niedokładne. W związku z powyższym, w omawianym tu eksperymencie wszystkie defekty traktowano jednakowo. Nie brano pod uwagę ich istotności.

4.3.2 Walidacja zewnętrzna

W omawianym w tym rozdziale eksperymencie badano zbiór bardzo różnorodnych projektów. Projekty te pochodzą z różnych branż, były wytwarzane w różnych procesach wytwarzania oprogramowania oraz reprezentują różne modele własności kodu źródłowego (otwarty oraz przemysłowy). Nie zmienia to jednak faktu, że przebadane projekty mogą mieć cechy wspólne. Z pewnością taką cechą wspólną jest język programowania, w którym były wytwarzane. Wszystkie badane projekty zostały napisane

w języku Java (co z drugiej strony eliminuje problem niejednoznacznych definicji metryk pomiędzy różnymi językami programowania). Jak pokazują badania dotyczące predykcji międzyprojektowej [46], [47], [48], [119], różnice pomiędzy różnymi projektami, analizowanymi z punktu widzenia predykcji defektów mogą być bardzo duże. W związku z tym trudno jednoznacznie wytyczyć granice, do których można posunąć generalizację uzyskanych wyników. Bez wątpienia można jednak stwierdzić, że dalsze badania dotyczące innych projektów poszerzą walidację zewnętrzną uzyskanych wyników.

4.3.3 Walidacja konkluzji

Wyniki eksperymentu są efektem testowania hipotez statystycznych na poziomie istotności $\alpha = 0,05$ oraz wyliczania siły efektu. Poszczególne badania były niezależne, więc nie było powodów do stosowania korekty Bonferroniego. Do obliczeń statystycznych stosowano pakiet Statistica i posługiwano się procedurami zaproponowanymi w [99], [100] i [101].

4.3.4 Walidacja konstrukcji

Na etapie przygotowywania danych do eksperymentu trzeba było rozwiązać problem odtworzenia powiązania pomiędzy defektami a klasami. Do tego celu wykorzystywano informacje pochodzące z systemów śledzenia defektów oraz z systemów kontroli wersji, co jest powszechnie przyjętą praktyką [26], [42], [120]. Jak jednak również powszechnie wiadomo [26], [35], [112], [120] podejście takie wiąże się z pewnymi problemami. Najpierw należy w systemie śledzenia defektów odróżnić te zgłoszenia, które reprezentują defekty od tych, które są powiązane z opracowywaniem nowych funkcjonalności. To czasami może być trudnym zadaniem i może stać się źródłem pomyłek [112]. Następnie defekty te są łączone z klasami na podstawie komentarzy z systemu kontroli wersji. Wykonanie tego manualnie dla dużych projektów nie jest możliwe, natomiast wykonanie tego automatycznie wiąże się z możliwością wystąpienia błędów [35]. Tu posłużono się metodą zautomatyzowaną, mianowicie wykorzystano opracowany do tego celu programem *BugInfo*. Program ten został szczegółowo opisany w Rozdziale 3.4.

Reguły określające co należy interpretować jako defekt mogą się różnić pomiędzy projektami. Antoniol i in. [1] pokazali, analizując projekty *Eclipse*, *JBoss* oraz *Mozilla*, że znaczna część zgłoszeń z systemu śledzenia defektów nie reprezentuje defektów. Kompletna eliminacja tych zgłoszeń z danych wykorzystanych w opisanym w tym rozdziale eksperymencie nie była możliwa z uwagi na zbyt dużą liczbę analizowanych projektów. Ograniczono się jedynie do automatycznej eliminacji tych zgłoszeń, które były w systemie śledzenia defektów zaklasyfikowane do jednej z kategorii używanych zazwyczaj do obsługi zgłoszeń nie związanych z defektami. Działanie takie nie gwarantuje niestety wyeliminowania wszystkich zgłoszeń nie będących defektami.

Istotny problem sprawiają również klasy anonimowe i wewnętrzne. Język programowania Java wymaga aby klasy takie były definiowane w tym samym pliku co klasa zewnętrzna. Z drugiej natomiast strony systemy kontroli wersji (*SubVersion*, *CVS*) działają na poziomie pliku. Trudno zatem rozstrzygnąć czy defekt odnotowany w logach systemu kontroli wersji pochodzi z klasy wewnętrznej czy zewnętrznej. Problemy te spowodowały, że zaleca się ignorowanie tych klas [1], [16], [17]. Istotność tego problemu jest bardzo łatwa do oszacowania, ponieważ liczba klas anonimowych i wewnętrznych jest możliwa do wyliczenia. W danych wykorzystanych do opisanego w tym rozdziale eksperymentu klasy te stanowiły $8,84\%$ wszystkich klas.

Źródłem podobnych problemów są operacje takie jak zmiana nazwy klasy oraz przeniesienie klasy pomiędzy pakietami. Operacje te powodują przerwanie historii klasy i tym samym uniemożliwiają zidentyfikowanie powiązania pomiędzy stanem klasy przed i po operacji. W efekcie powoduje to, że po

takiej operacji klasa jest interpretowana jako nowa, co ma negatywny wpływ zarówno na poprawność identyfikowania powiązań takiej klasy z defektami jak i na związane z nią metryki procesu.

Defekty były przypisywane do zgłoszeń na podstawie daty naprawienia zgłoszenia. Rozwiązanie takie było podyktowane czynnikami natury technicznej. Niezbędny do zidentyfikowania defektu komentarz z systemu śledzenia defektów jest wprowadzany dopiero w chwili zatwierdzania zmian rozwiązujących problem. Zdecydowanie ciekawszy byłby eksperyment, w którym udałoby się defekty łączyć z tymi wydaniami projektów, w których defekty te były wprowadzane.

W opisanym w tym rozdziale eksperymencie stosowano jedną konkretną metodę konstrukcji modeli predykcji defektów, mianowicie postępującą regresję liniową. W związku z tym trudno zawyrokować na ile wyniki tu uzyskane mają zastosowanie do eksperymentów, w których stosuje się inne metody konstrukcji modeli. Z drugiej jednak strony zastosowana metoda jest najmniej skomplikowana pod względem struktury uzyskiwanego modelu. Jeżeli więc możliwe jest generalizowanie wyników uzyskanych dla jednej metody na pozostałe, to zdecydowanie większe szanse na powodzenie ma generalizacja wychodząca od metod prostych (takich jak tu zastosowana postępująca regresja liniowa) niż od skomplikowanych (np. sieci neuronowe, regresja logistyczna). Wydaje się zatem, że dobór metody konstrukcji modelu predykcji defektów był optymalny pod kątem możliwości uogólniania uzyskanych wyników. Niemniej rozstrzygnięcie kwestii możliwości generalizowania wyników na inne metody wymaga replikacji eksperymentu z wykorzystaniem innej metody konstruowania modelu predykcji defektów.

4.4 Podsumowanie

Tabela 4.28: Ocena metryk

Metryka	Metoda I	Metoda II
Number of Revisions (NR)	–	–
Number of Distinct Commiters (NDC)	↗	–
Number of Modified Lines (NML)	↗	–
Is New (IN)	nie ma zastosowania	↘
Number of Defects in Previous Version (NDPV)	–	–
Number of Evening Revisions (NER)	↗?	?
Number of Pre–code–freeze Revisions (NPR)	?	?
Weighted Methods per Class (WMC)	nie ma zastosowania	–
Lines of Code (LOC)	nie ma zastosowania	↗
Kombinacja czterech metryk procesu	–	nie ma zastosowania

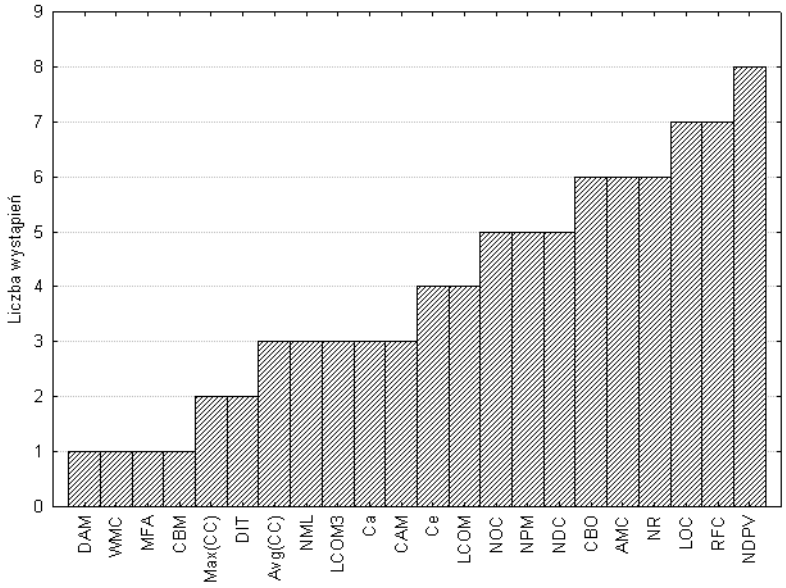
'↗' – istotna statystycznie poprawa predykcji;
'↘' – pogorszenie predykcji;
'–' – różnica nie była istotna statystycznie;
'?' – zbyt mała próbka by uzyskać wyniki istotne statystycznie.

Przeanalizowano wpływ dziewięciu różnych czynników na dokładność predykcji dostarczanej przez modele predykcji defektów. Siedem z przeanalizowanych czynników to metryki procesu. Tylko dwa spośród wszystkich analizowanych czynników były analizowane dla tak małej liczby projektów programistycznych, że utrudniało to analizę statystyczną uzyskanych wyników. Analizowano dwie różne metody wprowadzania metryki do modelu. Podsumowanie uzyskanych wyników umieszczono w Tabeli 4.28.

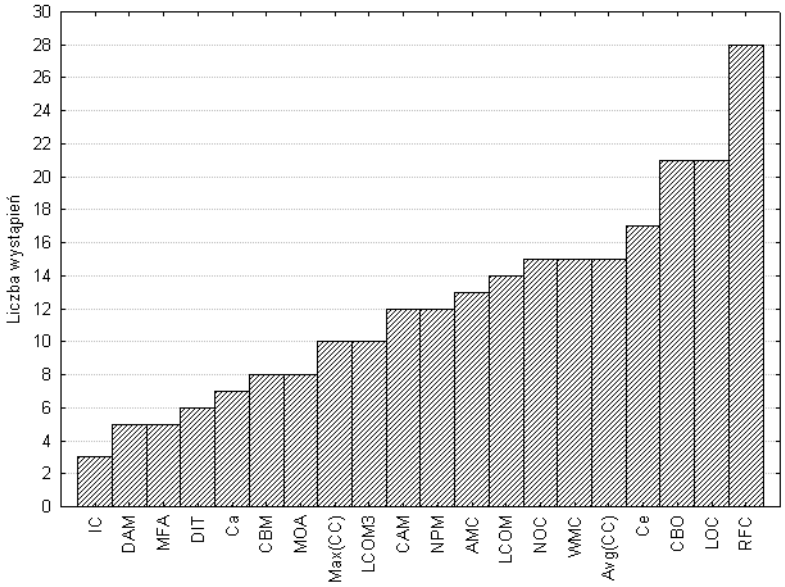
Wprowadzenie do modelu metryk *NDC*, *NML* (obu metodą I) oraz *LOC* (metodą II) dało w efekcie statystycznie istotną poprawę dokładności predykcji. Dla modeli, w których wprowadzenie którejś z powyższych metryk dało poprawę, wyliczono również siłę efektu. Zgodnie ze wzorcem zaproponowanym przez Kampenensa [56] uzyskano słaby efekt dla modeli wykorzystujących metryki *NML* oraz *LOC*. Natomiast dla modeli wykorzystujących metrykę *NDC* średni efekt. W świetle powyższych wyników nie ma większych wątpliwości co do użyteczności metryki *NDC* wprowadzanej metodą I. Jej wartość potwierdzają zarówno testy statystyczne jak i siła efektu. W przypadku metryk *NML*, *LOC* wyniki nie są już tak jednoznaczne. Uzyskano słaby efekt, ale mimo tego wyniki były istotne statystycznie na poziomie $\alpha = 0,05$. W związku z tym użyteczność tych metryk jest trudna do zanegowania i w sytuacji, w której metryka *NDC* nie jest dostępna, warto skorzystać z którejś z tych dwóch metryk.

Dla metryk *NR*, *NDPV*, *NER*, *NPR* oraz *WMC* nie uzyskano wyników istotnych statystycznie. Jedynie w przypadku metryk *NER* oraz *NPR* fakt ten można tłumaczyć małym rozmiarem próbki. W przypadku pozostałych metryk próbka była stosunkowo duża, a brak istotnych statystycznie wyników jest efektem małej zmiany dokładności predykcji. Spośród tych metryk najlepsze wyniki uzyskiwano dla metryki *NR*. Średnia różnica w ocenie modeli wykorzystujących tą metrykę i modeli referencyjnych wyniosła $5,05\%$, a w przypadku mediany aż $8,99\%$. Prawdopodobieństwo testowe, przy weryfikacji hipotezy dotyczącej użyteczności metryki *NR*, wyniosło $p = 0,075$, co jest wartością bliską zastosowanemu poziomowi istotności. Można więc, z pewną dozą ostrożności, zarekomendować metrykę *NR*, przynajmniej w takich sytuacjach, gdy żadna z dających lepsze wyniki metryk nie jest dostępna.

Metryki *NER* oraz *NPR* były analizowane w zaledwie kilku wydaniach projektu. W związku z tym nie było wystarczających danych do zweryfikowania hipotez dotyczących tych metryk. Niemniej poczynione obserwacje pozwalają wyciągnąć wstępne wnioski dotyczące tych metryk. Wprowadzanie metryki *NER* do modelu metodą I pozwoliło na uzyskanie ocen lepszych średnio o $2,70\%$. W przypadku mediany różnica ta wyniosła aż $10,25\%$. Wyniki takie są jak najbardziej zachęcające i skłaniają do przebadania tej metryki w kolejnych projektach. Zupełnie inne wyniki uzyskano dla metryki *NPR*.



Rysunek 4.1: Częstość występowania metryk w modelach z kombinacją czterech metryk procesu.



Rysunek 4.2: Częstość występowania metryk w modelach prostych.

Wprowadzanie jej do modelu spowodowało pogorszenie dokładności predykcji. Wynik taki stawia pod znakiem zapytania zasadność dalszych badań nad tą metryką.

Istotne statystycznie wyniki uzyskano dla metryki *IN*. Wprowadzenie tej metryki spowodowało jednak pogorszenie a nie poprawienie dokładności predykcji. Istnieją zatem przesłanki do sformułowania rekomendacji odradzającej stosowanie tej metryki w modelach predykcji defektów.

W przeprowadzanych eksperymentach badano wpływ każdej z metryk z osobna. Dodatkowo przeanalizowano również kombinację czterech metryk wprowadzanych do modelu predykcji defektów łącznie. Metryki te to *NR*, *NDC*, *NML*, *NDPV*. Są to metryki procesu wyliczone dla największej liczby wydań projektów. Spośród tych czterech metryk dwie (*NDC* i *NML*) wprowadzane z osobna poprawiały dokładność predykcji w sposób istotny statystycznie. Okazało się, że wprowadzenie kombinacji metryk dało poprawę predykcji na tyle niewielką, że nie była ona istotna statystycznie. Wyniki takie wskazują, że proces selekcji metryk zawsze powinien być przeprowadzony bardzo ostrożnie, gdyż wykorzystywanie zbyt dużej liczby metryk może czasami pogorszyć uzyskiwane wyniki.

Pomimo faktu, że modele wykorzystujące kombinację metryk nie okazały się być istotnie lepsze od modeli prostych, to ich budowa wewnętrzna może być bardzo ciekawa z uwagi na dużą liczbę wykorzystywanych metryk. Na Rysunku 4.1 pokazano ile razy poszczególne metryki były wykorzystywane w różnych modelach. Z uwagi na zastosowanie postępującej regresji, poszczególne modele wykorzystywały nie dość, że różne metryki to jeszcze różną ich liczbę. Jak można na Rysunku 4.1 zauważyć, najczęściej wykorzystywaną metryką była *NDPV*. Co jest szczególnie ciekawe z uwagi na fakt, że metryka ta nie została wskazana we wcześniejszych eksperymentach jako poprawiająca jakość predykcji. Metryki, które dokładność predykcji poprawiały, były wykorzystywane zdecydowanie rzadziej. Metryka *NDC* została użyta w pięciu modelach, a metryka *NML* w zaledwie trzech. W eksperymencie tym skonstruowano łącznie 12 modeli.

Na Rysunku 4.2 przedstawiono częstość występowania metryk w poszczególnych modelach prostych. Wyniki te dotyczą więc tylko metryk produktu, a są efektem skonstruowania łącznej liczby 51 modeli. Zdecydowanie najczęściej wykorzystywaną metryką jest tu *RFC*, nieco rzadziej używane były metryki *CBO* oraz *LOC*.

W Rozdziale 3 przedstawiono statystyki opisowe metryk. Między innymi podano tam również korelacje poszczególnych metryk z liczbą defektów. Ciekawą obserwacją jest rozbieżność pomiędzy wartościami korelacji a ocenami użyteczności metryk uzyskanymi z opisanych w tym rozdziale eksperymentów. Najmocniej skorelowane z liczbą defektów, spośród metryk procesu, były *NR* oraz *NDPV*. Natomiast eksperymenty pokazały, że metryki *NDC* oraz *NML* poprawiły dokładność predykcji w sposób istotny statystycznie. Wynik taki pokazuje, że ograniczanie analizy do zbadania korelacji jest zdecydowanie niewystarczające w pracach dotyczących predykcji defektów. Wyniki uzyskane z analizy korelacji mogą się nie potwierdzić podczas konstrukcji modelu, w na przykład wyniku występowania kolinearności pomiędzy zmiennymi.

Rozdział 5

Międzyprojektowa predykcja defektów

W rozdziale tym opisano eksperymenty dotyczące identyfikacji klastrów projektów z punktu widzenia predykcji defektów. Klaster projektów powinien mieć taką charakterystykę, aby do wszystkich projektów wchodzących w jego skład można było zastosować ten sam model predykcji defektów. Zdefiniowanie klastrów o takiej charakterystyce pozwoli na dokonywanie predykcji międzyprojektowej. Wystarczy bowiem zaklasyfikować projekt do odpowiedniego klastra i wtedy do predykcji w tym projekcie będzie można zastosować model pochodzący z klastra, czyli wytrenowany na innych projektach.

Identyfikację klastrów zaczęto od badania wstępnego, w którym założono a priori istnienie trzech klastrów [46] oraz [48]. W badaniu tym klastry definiowano na podstawie modelu własności kodu w projekcie. Mianowicie założono istnienie klastra projektów przemysłowych, projektów otwartych oraz projektów studenckich.

W drugim, bardziej zaawansowanym badaniu [47], nie czyniono żadnych założeń co do postaci klastrów. Zamiast tego do identyfikacji klastrów posłużono się metodami eksploracji danych. Wykorzystano metodę k–średnich, oraz sieć neuronową Kohonena.

W obu badaniach do podjęcia decyzji o tym, czy zidentyfikowany klaster rzeczywiście istnieje wykorzystywano metody statystyczne. Porównywano dokładność predykcji modelu pochodzącego z klastra z modelami wytrenowanymi na innych danych. Jeżeli model pochodzący z klastra dostarczał predykcji istotnie lepszych przyjmowano, że klaster istnieje.

5.1 Definicja eksperymentu

Istnieją istotne różnice pomiędzy strukturą eksperymentu wstępnego, badającego klastry zdefiniowane a priori, w oparciu o model własności kodu a eksperymentem wykorzystującym metody eksploracji danych do identyfikacji klastrów. W związku z tym struktury tych eksperymentów zostaną omówione w osobnych podrozdziałach.

5.1.1 Eksperyment wstępny

W ramach tego eksperymentu założono, że istnieją trzy klastry projektów z punktu widzenia predykcji defektów: projekty przemysłowe, projekty otwarte oraz projekty studenckie. Następnie weryfikowano istnienie każdego z tych klastrów. W tym celu konstruowano model dla badanego klastra, czyli taki, który był uczony na podstawie danych pochodzących ze wszystkich projektów należących do badanego klastra. Dalej konstruowano osobne modele dla każdego z dwóch pozostałych klastrów projektów oraz model bazujący na wszystkich projektach, które do badanego klastra nie należały. Tym sposobem

otrzymywano cztery różne modele, z których jeden został zbudowany na podstawie danych z badanego klastra, a trzy pozostałe na podstawie danych spoza badanego klastra. Aby zweryfikować istnienie badanego klastra, każdy z tych czterech modeli był stosowany do wykonania predykcji defektów w każdym z wydań projektów należących do badanego klastra. Decyzję o istnieniu klastra podejmowano na podstawie dokładności uzyskanych w ten sposób predykcji. Jeżeli badany klaster istnieje to model zbudowany dla tego klastra powinien dostarczać lepszych predykcji niż pozostałe modele.

Omawiany tu eksperyment można zdefiniować również w sposób bardziej formalny. W tym celu przyjmijmy, że W_P to zbiór wszystkich wydań projektów przemysłowych, W_O to zbiór wszystkich wydań projektów otwartych, W_S to zbiór wszystkich wydań projektów studenckich, $W_{\sim P}$ to zbiór wszystkich wydań projektów nienależących do klastra projektów przemysłowych, $W_{\sim O}$ to zbiór wszystkich wydań projektów nienależących do klastra projektów otwartych, a $W_{\sim S}$ to zbiór wszystkich wydań projektów nienależących do klastra projektów studenckich. Identyczną procedurę stosowano podczas weryfikowania istnienia każdego z postulowanych klastrów. W związku z tym procedura zostanie opisana na przykładzie weryfikowania klastra projektów przemysłowych. Weryfikacja pozostałych dwóch była analogiczna.

Podczas weryfikacji istnienia klastra projektów przemysłowych istotne są zbiory W_P, W_O, W_S oraz $W_{\sim P}$. Niech zatem w_P będzie wydaniem projektu należącym do W_P, w_O będzie wydaniem projektu należącym do W_O, w_S będzie wydaniem projektu należącym do W_S, a $w_{\sim P}$ wydaniem projektu należącym do $W_{\sim P}$.

Niech M_W będzie modelem predykcji defektów zbudowanym na podstawie wszystkich wydań projektów należących do zbioru W. W szczególności istnieją M_{W_P}, M_{W_O}, M_{W_S} oraz $M_{W_{\sim P}}$. Niech $E(M_W, w)$ będzie oceną (ewaluacją) dokładności predykcji modelu M_W w dokonywaniu predykcji w wydaniu w.

Funkcja $E(M, w)$ dokonująca oceny jest zdefiniowana identycznie jak w Rozdziale 4. Niech c_1, c_2, ..., c_n reprezentują klasy z wydania w ułożone w porządku malejącym według predykcji uzyskanych z modelu M. Niech d_1, d_2, ..., d_n będą liczbami rzeczywistych defektów występujących w kolejnych klasach. Niech D_j będzie liczbą defektów w j początkowych klasach: $D_j = \sum_{i=1}^{j} d_i$. Niech k będzie najmniejszym indeksem takim, że $D_k > 0,8 * D_n$, wtedy ocenę modelu będzie można wyliczyć z następującego wzoru:

$$E(M, w) = \frac{k}{n} \times 100\% \tag{5.1}$$

Skonstruowane wcześniej modele M_{W_P}, M_{W_O}, M_{W_S} oraz $M_{W_{\sim P}}$ zastosowano do wykonania predykcji w każdym z wydań projektów należących do zbioru projektów przemysłowych. Czyli dla każdego $w : w \in W_P$ wyliczono:

- $E(M_{W_P}, w)$ – zbiór uzyskanych wartości nazwijmy E_{W_P},
- $E(M_{W_O}, w)$ – zbiór uzyskanych wartości nazwijmy E_{W_O},
- $E(M_{W_S}, w)$ – zbiór uzyskanych wartości nazwijmy E_{W_S},
- $E(M_{W_{\sim P}}, w)$ – zbiór uzyskanych wartości nazwijmy $E_{W_{\sim P}}$.

Dysponując tak zdefiniowanymi zbiorami można sformułować następujące hipotezy zerowe:

- $H_{0,P,O}$ – E_{W_P} oraz E_{W_O} pochodzą z tego samego rozkładu. Odrzucenie tej hipotezy będzie oznaczało, że E_{W_P} i E_{W_O} pochodzą z różnych rozkładów.
- $H_{0,P,S}$ – E_{W_P} oraz E_{W_S} pochodzą z tego samego rozkładu.
- $H_{0,P,\sim P}$ – E_{W_P} oraz $E_{W_{\sim P}}$ pochodzą z tego samego rozkładu.

Hipotezy alternatywne:

- $H_{1,P,O}$ – E_{W_P} oraz E_{W_O} pochodzą z różnych rozkładów. Jeżeli dodatkowo E_{W_P} jest mniejsze od E_{W_O} to oznacza to, że stosowanie modelu zbudowanego na podstawie wersji projektów należących do W_O do dokonywania predykcji liczby defektów w wersjach projektów należącej do W_P, daje statystycznie gorsze rezultaty niż stosowanie modelu zbudowanego na podstawie wersji projektów należących do W_P.

- $H_{1,P,S}$ – E_{W_P} oraz E_{W_S} pochodzą z różnych rozkładów.

- $H_{1,P,\sim P}$ – E_{W_P} oraz $E_{W_{\sim P}}$ pochodzą z różnych rozkładów.

W celu zweryfikowania hipotez stosowano test t dla prób zależnych. Stosowanie tego testu wymaga spełnienia kilku założeń. Test ten mianowicie wymaga: aby badane wielkości były mierzone przynajmniej na skali interwałowej, aby dokonywane obserwacje były niezależne oraz aby badane próbki miały homogeniczną wariancję a przyjmowane przez nie wartości miały rozkład normalny. Dwa pierwsze założenia były spełnione we wszystkich przypadkach. Założenie o homogeniczności wariancji było sprawdzane przy pomocy testu Levenea, a normalność rozkładu przy pomocy testu Shapiro-Wilka. Zarówno homogeniczność wariancji jak i normalność były sprawdzane na poziomie istotności $\alpha = 0,05$. W żadnym z analizowanych przypadków założenia testu nie były naruszone. W związku z tym można było za każdym razem stosować test t dla prób zależnych. Hipotezy były testowane na poziomie istotności $\alpha = 0,05$.

5.1.2 Eksperyment wykorzystujący eksplorację danych

Istnieją podobieństwa pomiędzy strukturą eksperymentu wykorzystującego metody eksploracji danych a strukturą eksperymentu wstępnego. Jednak z uwagi na wystąpienie kilku różnic rzutujących na interpretację uzyskanych wyników struktura ta została tu w całości opisana.

W eksperymencie tym najpierw wyliczono dla każdego z wydań każdego z badanych projektów wektor korelacji. Wektor taki składał się z korelacji Pearsona pomiędzy wartościami poszczególnych metryk a liczbami defektów. Dodatkowo w wektorze korelacji umieszczono dwie wartości z korelacją niezwiązane. Mianowicie liczbę defektów przypadających średnio na klasę w danym wydaniu (DPC) oraz liczbę defektów przypadających średnio na klasę obarczoną przynajmniej jednym defektem (DPDC). Wektory te wykorzystano do automatycznego zidentyfikowania klastrów projektów. Do tego celu wykorzystywano metodę k-średnich oraz sieć neuronową Kohonena. Metoda k-średnich wymaga założenia dotyczącego liczby oczekiwanych klastrów. Liczbę klastrów oszacowano graficznie na podstawie dendrogramu. W celu zweryfikowania słuszności podziału na klastry stosowano analizę dyskryminacyjną. Dla każdego ze zidentyfikowanych klastrów z osobna stosowano procedurę weryfikującą jego istnienie z punktu widzenia predykcji defektów. Procedura ta polegała na skonstruowaniu dwóch modeli predykcji defektów. Pierwszy z nich był konstruowany przy użyciu tylko i wyłącznie danych pochodzących z wydań projektów należących do badanego klastra. Drugi model konstruowano korzystając ze wszystkich wydań wszystkich projektów. Tak utworzone modele używano do predykcji liczby defektów w każdym z wydań projektów należących do badanego klastra. Jeżeli model skonstruowany w oparciu o wydania należące do klastra dawał predykcje o tyle dokładniejsze, że różnica była istotna statystycznie, to przyjmowano, że badany klaster istnieje z punktu widzenia predykcji defektów.

Bardziej formalnie można tą procedurę opisać w następujący sposób. Załóżmy, że W to zbiór wszystkich wydań wszystkich projektów, a w to pojedyncze wydanie. Wtedy K jest zbiorem wydań

projektów zawierających wszystkie wydania w, które zostały zaklasyfikowane do badanego aktualnie klastra. K jest zatem podzbiorem W ($K \subset W$).

Istnieją dwa modele predykcji defektów: M_W oraz M_K. M_W to model ogólny, utworzony w oparciu o wszystkie analizowane wydania projektów, natomiast M_K to model zbudowany w oparciu tylko o wydania projektów należące do badanego klastra. Niech funkcja $E(M, w)$ będzie funkcją oceniającą dokładność predykcji modelu M w wydaniu w. Funkcja ta została zdefiniowana w Rozdziale 5.1.1, a jej wartość można wyliczyć ze wzoru 5.1.

Dla każdego w takiego, że $w \in K$ wyliczono:

- $E(M_W, w)$ – zbiór uzyskanych wartości nazwijmy E_W,
- $E(M_K, w)$ – zbiór uzyskanych wartości nazwijmy E_K.

Dysponując tak zdefiniowanymi zbiorami można sformułować następują hipotezę zerową:

- H_0 – E_W oraz E_K pochodzą z tego samego rozkładu.

Odrzucenie tej hipotezy będzie oznaczało przyjęcie następującej hipotezy alternatywnej:

- H_0 – E_W i E_K pochodzą z różnych rozkładów.

Jeżeli nie dość że zostanie przyjęta hipoteza alternatywna, to jeszcze dodatkowo średnia z E_K będzie mniejsza od średniej z E_W (mniejsza wartość oceny oznacza dokładniejsze predykcje). To będzie to oznaczało, że stosowanie modelu zbudowanego na podstawie wersji projektów należących do W do dokonywania predykcji liczby defektów w wersjach projektów należącej do K daje statystycznie gorsze rezultaty niż stosowanie modelu zbudowanego na podstawie wersji projektów należących do K, czyli do badanego klastra.

W celu zweryfikowania hipotez stosowano test t dla prób zależnych. Stosowanie tego testu wymaga spełnienia kilku założeń. Test ten mianowicie wymaga: aby badane wielkości były mierzone przynajmniej na skali interwałowej, aby dokonywane obserwacje były niezależne i aby badane próbki miały homogeniczną wariancję a przyjmowane przez nie wartości miały rozkład normalny. Dwa pierwsze założenia były spełnione we wszystkich przypadkach. Założenie o homogeniczności wariancji było sprawdzane przy pomocy testu Levenea, a normalność rozkładu przy pomocy testu Shapiro-Wilka. Zarówno homogeniczność wariancji jak i normalność były sprawdzane na poziomie istotności $\alpha = 0,05$. Jeżeli któreś z założeń nie było spełnione to stosowano test Wilcoxona dla par obserwacji. Hipotezy były testowane na poziomie istotności $\alpha = 0,05$.

Aby zweryfikować poprawność przeprowadzonej klasteryzacji wykonywano analizę dyskryminacyjną. Wykorzystano w tym celu statystykę lambda Wilksa, która jest standardową statystyką wykorzystywaną do wyznaczania istotności statystycznej mocy dyskryminacyjnej [101]. Aby sprawdzić, czy więcej niż k funkcji ma nieistotną miarę dyskryminacyjną, stosuje się statystykę Λ.

$$\Lambda = \prod_{i=k+1}^{q} \frac{1}{1+\lambda_i} \tag{5.2}$$

Statystykę tą wylicza się ze wzoru 5.2, gdzie q to maksymalna liczba funkcji dyskryminacyjnych; λ_i to i–ta wartość własna; k to liczba klastrów.

Analiza dyskryminacyjna pozwala na rozstrzygnięcie, które zmienne najlepiej dzielą dany zbiór przypadków na zidentyfikowane klastry. Zmienne używane do rozróżniania klastrów nazywane są w tej analizie zmiennymi dyskryminującymi. Analiza ta pozwala również ocenić na ile istotna jest różnica

pomiędzy klastrami. Analizę dyskryminacyjną warto zacząć od wyznaczenia kanonicznych funkcji dyskryminacyjnych rozdzielających klastry. Do tego celu stosuje się funkcje liniowe postaci:

$$D_{kj} = \beta_0 + \beta_1 x_{1kj} + ... + \beta_p x_{pkj} \tag{5.3}$$

gdzie: p to liczba zmiennych dyskryminacyjnych; n to liczebność próby; g to liczba klastrów; D_{kj} to wartość kanonicznej dyskryminacyjnej funkcji dla k–tego przypadku w j–tym klastrze; x_{ikj} – wartość i–tej zmiennej dyskryminacyjnej dla k–tego przypadku w j–tym klastrze; β_i to współczynnik kanonicznej funkcji dyskryminacyjnej; $k = 1, ..., n$; $j = 1, ..., g$; $i = 1, ..., p$. Wartości współczynników β_i są wyznaczane w taki sposób aby zminimalizować zmienność wewnątrzklastrową oraz równocześnie zmaksymalizować zmienność międzyklastrową. Zmienności te można opisać przy pomocy macierzy międzyklastrowych i wewnątrzklastrowych sum kwadratów odchyleń od średniej. Problem znalezienia współczynników β_i sprowadza się wtedy do rozwiązania równania macierzowego $(\mathbf{M} - \lambda\mathbf{W})\beta = \mathbf{0}$ gdzie: $\mathbf{M}$ to międzyklastrowa macierz kwadratów; $\mathbf{W}$ to wewnątrzklastrowa macierz kwadratów; β to wektor współczynników kanonicznych funkcji dyskryminacyjnych; λ to wartość własna. Rozwiązanie tego równania macierzowego prowadzi do wyznaczenia szeregu współczynników nazywanych wartościami własnymi, którym odpowiada szereg wektorów własnych stanowiących poszukiwane współczynniki kanonicznej funkcji dyskryminacyjnej. Uzyskane tym sposobem funkcje pozwalają już na wyliczenie wspomnianej wcześniej statystyki lambda Wilksa. Wartości Λ mieszczą się w przedziale od 0 (perfekcyjna moc dyskryminacyjna analizowanej klasteryzacji) do 1 (żadna moc dyskryminacyjna). Statystyka ta ma w przybliżeniu rozkład chi–kwadrat i w związku z tym do testowania jej istotności stosuje się statystykę χ^2:

$$\chi^2 = -[n - \frac{p+g}{2} - 1] * ln\Lambda_k \tag{5.4}$$

mającą rozkład chi–kwadrat o $(p-k)(g-k-1)$ stopniach swobody. Wartość p oznacza tu liczbę zmiennych dyskryminacyjnych, g liczbę klastrów a k liczbę funkcji dyskryminacyjnych. Do oceny istotności statystycznej mocy dyskryminacyjnej w oparciu o statystykę Λ posłużono się poziomem istotności $\alpha = 0,05$.

Aby ocenić wkład poszczególnych zmiennych dyskryminujących w zidentyfikowany podział na klastry można przeprowadzić analizę dyskryminacyjną krokowo [101]. Wprowadza się wtedy (względnie usuwa) stopniowo kolejne zmienne dyskryminacyjne, a dla uzyskanych stanów pośrednich wyliczyć można lambdę Wilksa oraz cząstkową lambdę Wilksa (określa wkład danej zmiennej do dyskryminacji klastrów). Lambdę Wilksa można przekształcić na standardową statystykę F Snedecora, która to pozwala na wyliczenie prawdopodobieństwa testowego p. Uzyskane tym sposobem prawdopodobieństwo testowe p pozwala na ocenę istotności wkładu danej zmiennej dyskryminującej. Istotność wkładu poszczególnych zmiennych była oceniana na poziomie istotności $\alpha = 0,05$. Oprócz wkładu danej zmiennej ważna jest również jej redundancja z pozostałymi dostępnymi zmiennymi. Wielkość tą wyrażano przy pomocy tolerancji. Wartość tolerancji zmiennej równa jest 1 minus kwadrat korelacji tej zmiennej ze wszystkimi pozostałymi zmiennymi. Zatem niskie wartości tolerancji oznaczają wysoki poziom redundancji danej zmiennej z pozostałymi.

5.2 Wyniki eksperymentu

Wyniki podzielono na dwie części. W pierwszej opisany został eksperyment wstępny, w drugiej natomiast eksperyment wykorzystujący metody eksploracji danych do identyfikacji klastrów.

Tabela 5.1: Statystyki opisowe klastrów

	Klaster przemysłowy		Klaster otwarty		Klaster studencki	
Metryka	Średnia	σ	Średnia	σ	Średnia	σ
WMC	5,0833	8,5371	10,7079	15,8810	9,7185	10,7852
DIT	3,0732	1,7090	2,1186	1,3861	2,1084	1,6747
NOC	0,6068	9,0260	0,5203	3,2892	0,2304	1,4614
CBO	15,2448	21,0612	10,6039	17,4058	8,0209	8,1193
RFC	24,5002	27,1012	28,3554	36,6480	26,0250	30,6231
LCOM	37,2557	668,9347	109,5427	771,2874	62,0375	276,458
Ca	2,8127	17,9642	5,3722	15,6747	3,8446	6,6645
Ce	12,4918	10,3615	5,5802	7,4616	4,7018	5,8509
NPM	3,3430	7,8056	8,2898	12,0437	7,3796	8,6741
LCOM3	1,3623	0,5498	1,1048	0,6678	1,0920	0,6318
LOC	168,6867	364,3159	299,6146	725,9929	248,1272	463,429
DAM	0,1842	0,3382	0,5074	0,4666	0,5988	0,4680
MOA	0,0816	1,1093	0,7853	1,8602	0,7842	1,5888
MFA	0,6286	0,3926	0,4168	0,4166	0,3472	0,4261
CAM	0,5567	0,2302	0,4738	0,2503	0,4884	0,2497
IC	1,1109	1,0843	0,5319	0,8295	0,2503	0,4813
CBM	1,7190	2,3363	1,4832	3,0753	0,5339	1,4645
AMC	30,8714	40,9167	28,3855	77,2440	21,2490	24,7025
Max(CC)	3,1729	5,5626	4,1402	8,0528	3,2221	5,5328
Avg(CC)	1,2396	1,4443	1,3119	1,2212	1,1384	0,8232

5.2.1 Wyniki eksperymentu wstępnego

W eksperymencie tym założono istnienie trzech klastrów projektów. Przynależność do klastra wynikała z modelu własności kodu źródłowego projektu. Założono mianowicie istnienie klastra projektów przemysłowych, otwartych oraz studenckich. Wartości przyjmowane przez poszczególne metryki w klastrach zostały przedstawione w Tabeli 5.1. Można się tam doszukać różnic pomiędzy klastrami, jednak bez przeprowadzenia wnikliwej analizy trudno ocenić ich istotność. Większość metryk powiązanych z rozmiarem (WMC, LCOM, LOC, Max(CC) i Avg(CC)) przyjmuje największe wartości w projektach otwartych. Co jednak ciekawe, metryki opisujące powiązania już wcale największych wartości dla tej klasy projektów nie przyjmują – metryki CBO i Ce przyjmują największe wartości dla projektów przemysłowych.

Tabela 5.2: Statystyki opisowe – dopasowanie modeli do projektów przemysłowych

	Model nieprzemysłowy	Model otwarty	Model studencki	Model przemysłowy
Średnia	53,96	55,38	73,59	51,77
σ	13,29	12,01	9,68	9,76

Klaster projektów przemysłowych. Jak pokazuje Tabela 5.2 model skonstruowany na podstawie projektów przemysłowych dał najlepsze predykcje w zbiorze projektów przemysłowych. Był minimalnie lepszy od modelu skonstruowanego na podstawie projektów otwartych oraz od modelu skonstruowanego na podstawie wszystkich projektów nie należących do zbioru projektów przemysłowych. Zdecydowanie najgorszy okazał się być model skonstruowany na podstawie projektów studenckich. Jego

średni wynik jest równy 73, 59, co jest wynikiem bardzo słabym, jeżeli wziąć pod uwagę fakt, że model zupełnie losowy powinien uzyskiwać średni wynik bliski wartości 80.

Tabela 5.3: Normalność rozkładu – predykcje w projektach przemysłowych

	Model nieprzemysłowy	Model otwarty	Model studencki	Model przemysłowy
W	0,970	0,975	0,946	0,960
p	0,650	0,794	0,220	0,443

Na podstawie Tabeli 5.3 można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o normalności rozkładu w przypadku żadnego z analizowanych modeli. Tym samym oznacza to, że założenia testu t dotyczące normalności rozkładu nie są naruszone.

Tabela 5.4: Homogeniczności rozkładów – model przemysłowy a pozostałe

	Model nieprzemysłowy	Model otwarty	Model studencki
F(1,df), df=56	3,149	1,207	0,007
p	0,083	0,278	0,936

Analizę homogeniczności rozkładu wyników uzyskanych przez zastosowanie modelu przemysłowego z rozkładami wyników uzyskanych dzięki zastosowaniu pozostałych modeli zaprezentowano w Tabeli 5.4. W oparciu o te wyniki można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o homogeniczności rozkładów. Tym samym oznacza to, że założenia testu t nie są naruszone.

Tabela 5.5: Testowanie hipotez – predykcje w projektach przemysłowych

	$H_{0,P,\sim P}$	$H_{0,P,O}$	$H_{0,P,S}$
t, df=23	0,708	1,223	7,320
p	0,486	0,234	**0,000**

Dla modelu przemysłowego uzyskiwano najlepsze predykcje, ale mimo tego nie było podstaw do odrzucenia hipotez zerowych dotyczących modelu nieprzemysłowego oraz modelu otwartego. Różnice pomiędzy predykcjami dostarczanymi przez model przemysłowy a predykcjami dostarczonymi przez model otwarty i nieprzemysłowy była na tyle mała, że nie okazała się być istotną statystycznie. Model studencki dostarczał predykcji zdecydowanie gorszych. Różnica pomiędzy ocenami predykcji modelu studenckiego a ocenami predykcji modelu przemysłowego była istotna statystycznie. W związku z tym można odrzucić hipotezę zerową $H_{0,P,S}$ i przyjąć, że model studencki nie nadaje się do wykonywania predykcji w projektach przemysłowych. Wyniki testowania wszystkich hipotez znajdują się w Tabeli 5.5.

Tabela 5.6: Statystyki opisowe – dopasowanie modeli do projektów otwartych

	Model przemysłowy	Model nieotwarty	Model studencki	Model otwarty
Średnia	57,11	56,57	65,59	54,08
σ	19,12	17,67	14,22	16,85

Klaster projektów otwartych. Jak pokazuje Tabela 5.6 zdecydowanie najdokładniejsze predykcje dla projektów należących do klastra projektów otwartych uzyskano dla modelu otwartego. Był on lepszy o 2,5% od modelu nieotwartego, o 3% od modelu przemysłowego oraz aż o 11,5% od modelu studenckiego. Ponownie, podobnie jak w przypadku projektów przemysłowych, najgorsze predykcje zostały dostarczone przez model studencki.

Tabela 5.7: Normalności rozkładu – predykcje w projektach otwartych

	Model przemysłowy	Model nieotwarty	Model studencki	Model otwarty
W	0,956	0,971	0,983	0,982
p	0,116	0,380	0,768	0,734

Na podstawie Tabeli 5.7 można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o normalności rozkładu w przypadku żadnego z analizowanych modeli. Tym samym oznacza to, że założenia testu t dotyczące normalności rozkładu nie są naruszone.

Tabela 5.8: Homogeniczności rozkładów – model otwarty a pozostałe

	Model przemysłowy	Model nieotwarty	Model studencki
F(1,df), df=80	0,858	0,093	1,338
p	0,357	0,761	0,251

Na podstawie Tabeli 5.8 można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o homogeniczności rozkładu ocen uzyskanych dla modelu otwartego i rozkładów ocen uzyskanych dla pozostałych modeli. Tym samym oznacza to, że założenia testu t nie zostały naruszone.

Tabela 5.9: Testowanie hipotez – predykcje w projektach otwartych

	$H_{0,O,P}$	$H_{0,O,\sim O}$	$H_{0,O,S}$
t, df=40	2,091	1,907	4,405
p	**0,043**	0,064	**0,0001**

Jak pokazuje Tabela 5.9 predykcje uzyskane dla modelu otwartego były istotnie dokładniejsze w większości przypadków. Wynik taki pozwolił na odrzucenie hipotez zerowych: $H_{0,O,P}$ oraz $H_{0,O,S}$. W przypadku hipotezy $H_{0,O,\sim O}$ prawdopodobieństwo testowe minimalnie przekroczyło przyjęty poziom istotności $\alpha = 0,05$. Wynik taki oznacza, że predykcje uzyskane z modelu otwartego są zauważalnie lepsze od predykcji uzyskanych z modeli przemysłowego, studenckiego i nieotwartego gdy obiektem badań są projekty należące do klastra projektów otwartych. W przypadku modeli przemysłowego oraz studenckiego różnica ta był istotna statystycznie.

Tabela 5.10: Statystyki opisowe – dopasowanie modeli do projektów studenckich

	Model przemysłowy	Model otwarty	Model studencki	Model niestudencki
Średnia	56,34	50,60	55,02	53,19
σ	20,71	15,56	20,21	18,54

Klaster projektów studenckich. Oceny predykcji wykonanych dla projektów należących do klastra projektów studenckich zostały przedstawione w Tabeli 5.10. Najlepsze predykcje zostały uzyskana dla modelu otwartego, a najgorsze dla modelu przemysłowego. Model studencki okazał się być jedynie nieznacznie lepszy od dającego najgorsze predykcje, modelu przemysłowego. Różnica w ocenie tych dwóch modeli wyniosła $1,3\%$.

Tabela 5.11: Normalności rozkładu – predykcje w projektach studenckich

	Model przemysłowy	Model otwarty	Model studencki	Model otwarty
W	0,921	0,946	0,973	0,942
p	0,151	0,401	0,869	0,345

Na podstawie Tabeli 5.11 można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o normalności rozkładu w przypadku żadnego z analizowanych modeli. Tym samym oznacza to, że założenia testu t dotyczące normalności rozkładu nie zostały naruszone.

Tabela 5.12: Homogeniczności rozkładów – model przemysłowy a pozostałe

	Model przemysłowy	Model otwarty	Model niestudencki
F(1,df), df=32	0,271	1,343	0,303
p	0,606	0,255	0,586

Na podstawie Tabeli 5.12 można stwierdzić, że nie ma podstaw do odrzucenia hipotezy o homogeniczności rozkładu ocen uzyskanych dla modelu studenckiego i rozkładów ocen uzyskanych dla pozostałych modeli. Tym samym oznacza to, że założenia testu t nie zostały naruszone.

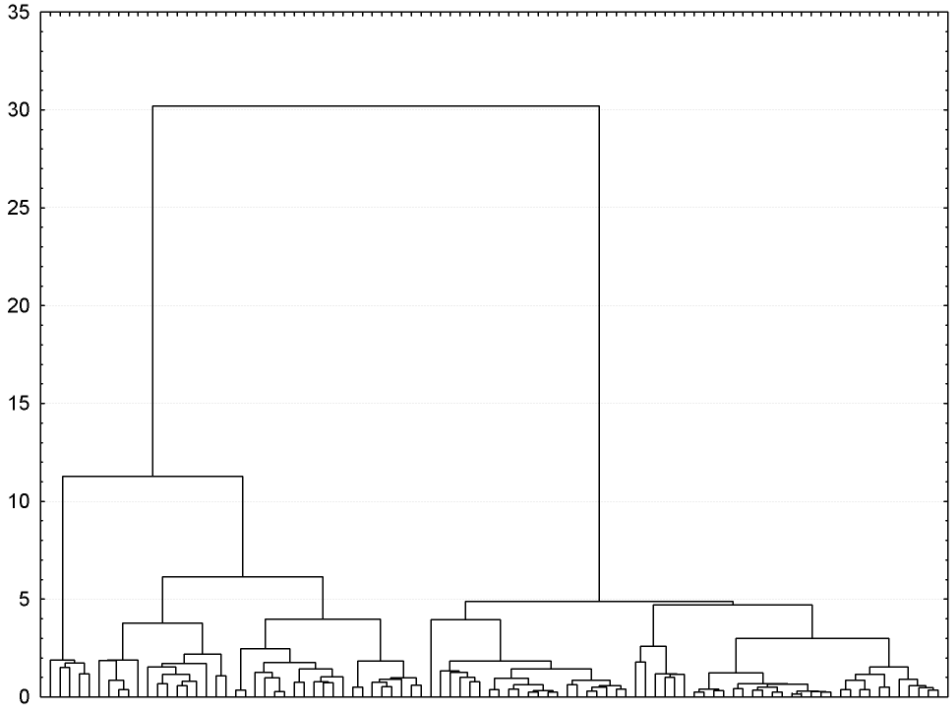
Tabela 5.13: Testowanie hipotez – predykcje w projektach studenckich

	$H_{0,S,P}$	$H_{0,S,O}$	$H_{0,S,\sim S}$
t, df=16	0,312	-1,484	-0,696
p	0,759	0,157	0,496

Jak pokazuje Tabela 5.13 w przypadku predykcji wykonywanych dla projektów studenckich żadna z hipotez zerowych nie może zostać odrzucona. Ujemna wartość statystyki t dla hipotez $H_{0,S,O}$ oraz $H_{0,S,\sim S}$ jest efektem uzyskania dla modelu studenckiego gorszych predykcji niż dla porównywanych w tych hipotezach modeli otwartego i niestudenckiego. Fakt, że prawdopodobieństwo testowe p jest w przypadku tych hipotez większe od przyjętego poziomu istotności $\alpha = 0,05$ wskazuje, że zastosowanie modelu studenckiego doprowadziło do takiego pogorszenia dokładności predykcji, którego nie można uznać za istotne statystycznie. Mimo tego uzyskanie tak słabych wyników dla tego modelu zdecydowanie zniechęca do jego dalszego stosowania.

5.2.2 Wyniki eksperymentu wykorzystujące eksplorację danych

Wyniki eksperymentu wykorzystującego metodę k-średnich oraz sieć neuronową Kohonena do identyfikowania klastrów zostały podzielone na trzy części. Wpierw opisany został podział na klastry i przedstawione zostały statystyki opisowe każdego z klastrów. Następnie zaprezentowany został rezultat zastosowania procedury weryfikującej istnienie poszczególnych klastrów. W końcowej części przedstawiona została analiza dyskryminacyjna badanych w tym rozdziale podziałów na klastry.



Rysunek 5.1: Dendrogram klasteryzacji

Statystyki opisowe klastrów W metodzie k-średnich należy z góry określić liczbę klastrów. Liczbę klastrów można próbować odgadywać, ale zdecydowanie lepiej do tego celu posłużyć się dendrogramem. Na Rysunku 5.1 przedstawiono dendrogram skonstruowany dla wszystkich badanych wydań projektów. Można na nim zobaczyć jak odległe od siebie są wektory korelacji wyliczone dla poszczególnych wydań projektów. W pierwszym przybliżeniu wyraźnie widać na dendrogramie podział na dwa klastry. Posuwając się niżej, w głąb wykresu można się doszukać podziału na trzy, cztery, pięć, a nawet sześć klastrów. Aby nie przesadzić z ziarnistością podziału (identyfikowanie bardzo małych klastrów nie dostarcza użytecznych informacji) ograniczono się do przeanalizowania podziału na dwa oraz na cztery klastry.

Klasteryzacja przy pomocy k–średnich. Wyniki podziałów na klastry przy pomocy metody k–średnich wyrażone przy pomocy statystyk opisowych metryk oprogramowania zostały przedstawione w Tabeli 5.14 oraz Tabeli 5.15. Klastry zidentyfikowane w ramach podziału na dwa klastry nazwano: "1 z 2" oraz "2 z 2". W podziale na dwa klastry widzimy znaczne różnice w średnich wartościach metryk: WMC, LCOM, Ca, Ce, NPM, LOC, DAM, MOA oraz IC. Natomiast niemal identyczne wartości średnie w obu klastrach przyjmują metryki: NOC, RFC, CAM, CBM, AMC oraz Avg(CC). W podziale na cztery klastry trudno wyciągnąć tego typu wnioski dotyczące różnic w wartościach poszczególnych metryk, ponieważ częstą sytuacją jest podobna wartość średnia metryki w dwóch klastrach i zupełnie inna w trzecim. Warto tu przy okazji zauważyć, że w Tabeli 5.15 przedstawiono jedynie statystyki dotyczące trzech klastrów. Czwarty ze zidentyfikowanych w ramach tego podziału klastrów składał się z zaledwie jednego wydania projektu, w związku z czym został on wykluczony z dalszej analizy i przez to pominięty w Tabeli 5.15. Klastry zidentyfikowane w ramach podziału na cztery klastry nazwano: "1 z 4", "2 z 4" oraz "3 z 4". Mimo wszystko istnieje jednak kilka metryk przyjmujących bardzo podobne wartości w każdym z klastrów: RFC, CAM, AMC, Avg(CC).

Tabela 5.14: Statystyki opisowe klastrów – podział na dwa klastry

	Klaster "1 z 2"		Klaster "2 z 2"	
Metryka	Średnia	σ	Średnia	σ
WMC	5,8734	9,6876	11,0142	16,9912
DIT	2,9102	1,7076	2,1428	1,4388
NOC	0,5797	8,2258	0,5335	3,6981
CBO	14,4807	20,3796	10,2336	17,1367
RFC	25,0114	28,5097	29,1586	38,0948
LCOM	46,1617	669,2989	108,3410	760,6433
Ca	3,1104	17,3781	5,5046	15,5938
Ce	11,4410	10,2667	5,4166	7,1537
NPM	4,0685	8,6658	8,1567	11,5771
LCOM3	1,3295	0,5765	1,0836	0,6426
LOC	183,6548	405,8519	321,7214	828,6884
DAM	0,2408	0,3835	0,4886	0,4560
MOA	0,1726	1,1867	0,8426	1,9522
MFA	0,5913	0,4065	0,4179	0,4099
CAM	0,5445	0,2344	0,4638	0,2456
IC	1,0110	1,0637	0,5411	0,8077
CBM	1,6761	2,4743	1,4808	2,8981
AMC	29,8496	45,0079	29,3778	79,8559
Max(CC)	3,3695	5,8117	4,4281	9,3875
Avg(CC)	1,2801	1,4374	1,3549	1,3001

Tabela 5.15: Statystyki opisowe klastrów – podział na cztery klastry

	Klaster "1 z 4"		Klaster "2 z 4"		Klaster "3 z 4"	
Metryka	Średnia	σ	Średnia	σ	Średnia	σ
WMC	5,9906	9,9215	10,0542	9,8694	11,3748	17,8869
DIT	2,8938	1,7053	1,9310	1,3334	2,1001	1,4189
NOC	0,5804	8,1383	0,4064	2,5666	0,5149	3,8053
CBO	14,3851	20,4307	10,8448	13,2450	9,9392	15,0004
RFC	25,1487	28,8219	29,2291	25,8156	28,7060	38,8128
LCOM	48,1630	683,7263	49,6281	239,7431	107,2025	606,5563
Ca	3,1833	17,4877	5,2069	12,9001	5,3736	12,9529
Ce	11,2899	10,2269	5,8399	5,5164	5,2714	7,5210
NPM	4,1599	8,7441	7,7365	8,8349	8,4450	12,0871
LCOM3	1,3226	0,5807	0,9048	0,5600	1,1049	0,6354
LOC	185,6050	414,4684	294,1059	345,9508	350,4394	914,4424
DAM	0,2488	0,3889	0,7182	0,4223	0,4470	0,4468
MOA	0,1887	1,2009	0,6133	1,3611	0,8835	2,1391
MFA	0,5877	0,4073	0,3720	0,4156	0,4061	0,4069
CAM	0,5427	0,2353	0,4661	0,2293	0,4585	0,2405
IC	0,9990	1,0609	0,4901	0,8035	0,5394	0,7962
CBM	1,6578	2,4668	1,2611	2,3446	1,7060	3,1924
AMC	29,6145	45,7203	27,3849	22,3357	33,0451	87,2619
Max(CC)	3,3959	5,9046	3,8571	4,4974	4,4975	9,9625
Avg(CC)	1,2812	1,4332	1,3661	0,9295	1,3691	1,3390

Klasteryzacja przy pomocy sieci Kohonena. Sieć neuronowa Kohonena jest algorytmem działającym niedeterministycznie. Efektem tego było otrzymywanie innego podziału na klastry przy każdym uruchomieniu sieci. W związku z tym podział na klastry, który został opisany przy pomocy metryk oprogramowania w Tabeli 5.16 jest efektem uśrednienia wyników kilkukrotnego przeprowadzenia klasteryazji przy pomocy sieci Kohonena. Za każdym razem stosowano sieć tak samo skonfigurowaną. Jako klastry uznano te zbiory wydań projektów, które składały się z wydań projektów klasyfikowanych najczęściej do tych samych klastrów przez sieć. Istnieje również kilka wydań projektów, które nie zostały zakwalifikowane do żadnego z klastrów. Wydania te w każdej analizie były klasyfikowane przez sieć do innego zbioru albo tworzyły zbiory jednoelementowe, czyli były przez sieć klasyfikowane do takiego zbioru, do którego nie został zakwalifikowany żaden inny projekt. Klastry zidentyfikowane przy pomocy sieci Kohonena nazwano: "przemysłowy A", "przemysłowy B", "przemysłowo–otwarty" oraz "otwarty". Brakuje tutaj klastra studenckiego. Jest to wynikiem klasyfikowania wydań projektów studenckich do różnych klastrów i w dużym stopniu potwierdza wyniki eksperymentu wstępnego. W przypadku podziału na klastry będącego efektem zastosowania sieci Kohonena, jak można zobaczyć w Tabeli 5.16, tylko dwie metryki mają bardzo podobne wartości średnie we wszystkich klastrach. Są to metryka CAM oraz metryka Avg(CC). Warto tu zauważyć, że obie te metryki przyjmowały podobne wartości również w klastrach skonstruowanych przy pomocy metody k–średnich.

Tabela 5.16: Statystyki opisowe klastrów – sieć Kohonena

	Klaster "przem. A"		Klaster "przem. B"		Klaster "przem.–otw."		Klaster "otwarty"	
Metryka	Śred.	σ	Śred.	σ	Śred.	σ	Śred.	σ
WMC	4,39	5,71	5,63	9,29	11,48	18,01	9,59	11,61
DIT	3,29	1,56	2,83	1,73	2,28	1,51	2,05	1,28
NOC	0,66	9,90	0,55	7,99	0,50	3,34	0,56	2,94
CBO	16,21	20,60	14,39	20,52	10,88	17,20	10,85	19,45
RFC	22,72	22,04	25,40	29,50	29,33	40,08	26,31	30,91
LCOM	16,98	201,61	44,71	721,55	132,36	932,74	79,13	479,53
Ca	2,51	17,89	2,97	17,30	5,56	14,63	5,51	18,64
Ce	13,72	9,99	11,50	10,39	6,03	8,36	5,69	6,40
NPM	2,80	4,88	3,82	8,51	8,84	13,31	7,48	9,70
LCOM3	1,36	0,53	1,35	0,56	1,17	0,66	1,03	0,67
LOC	143,04	266,51	181,93	389,64	375,56	871,94	201,81	372,17
DAM	0,12	0,28	0,23	0,37	0,42	0,45	0,62	0,46
MOA	0,09	1,56	0,10	0,85	0,86	2,14	0,73	1,40
MFA	0,68	0,35	0,57	0,41	0,45	0,43	0,40	0,40
CAM	0,57	0,20	0,54	0,24	0,47	0,25	0,47	0,25
IC	1,27	1,08	0,96	1,04	0,62	0,92	0,49	0,75
CBM	1,70	1,83	1,60	2,45	2,10	3,88	0,91	1,91
AMC	31,44	46,22	29,42	36,24	38,18	101,74	17,87	49,84
Max(CC)	2,74	4,11	3,54	6,24	4,50	8,57	3,32	6,64
Avg(CC)	1,16	1,30	1,32	1,51	1,39	1,37	1,14	0,86

Wyniki eksperymentu – podział na dwa klastry. Model skonstruowany na podstawie wydań projektów należących do klastra "1 z 2" (M_K) dostarczał predykcji minimalnie dokładniejszych niż model ogólny (M_W). Jak można zobaczyć w Tabeli 5.17 różnica ta wynosiła zaledwie $0,05\%$. W związku z tym, jak należało oczekiwać, różnica ta nie okazała się być istotna statystycznie i tym samym nie ma podstaw

do stwierdzenia, że klaster "1 z 2" istnieje z punktu widzenia predykcji defektów. Hipotezę testowano przy pomocy testu t, gdyż założenia dotyczące normalności rozkładu i homogeniczności wariancji nie zostały naruszone. Wyniki zaprezentowano w Tabeli 5.18.

Tabela 5.17: Klaster "1 z 2" – oceny modeli

	Średnia	σ
E_W	49,73	19,64
E_K	49,67	18,37

Tabela 5.18: Klaster "1 z 2" – testy hipotez

		E_W	E_K
Test	W	0,987	0,991
Shapiro–Wilka	p	0,782	0,931
Test	df	118	
Levenea	F(1,df)	0,434	
	p	0,511	
	t	0,057	
Test–t	df	60	
	p	**0,954**	

Model zbudowany na podstawie danych pochodzących z klastra "2 z 2" dostarczył gorszych predykcji niż model ogólny zbudowany na podstawie wszystkich dostępnych wydań projektów. Chodzi tu o predykcje dla wydań projektów należących do klastra "2 z 2". Z uwagi na ten fakt nie ma potrzeby stosowania testu statystycznego. Można od razu stwierdzić, iż nie ma podstaw do stwierdzenia że klaster "2 z 2" istnieje z punktu widzenia predykcji defektów. Oceny modeli zastosowanych do predykcji w wydaniach projektów należących do klastra "2 z 2" przedstawiono w Tabeli 5.19.

Tabela 5.19: Klaster "2 z 2" – oceny modeli

	Średnia	σ
E_W	47,18	17,80
E_K	47,41	17,29

Nie udało się wykazać istnienia ani klastra "1 z 2" ani klastra "2 z 2". W związku z tym można zakwestionować użyteczność dokonanego podziału na dwa klastry. Wynik taki pokazuje, że wydania wchodzące w skład każdego z klastrów z osobna były na tyle różnorodne, że model uśredniający ich charakterystyki nie był lepszy od modelu ogólnego, który uśredniał charakterystyki wszystkich dostępnych wydań projektów. Poprawę w tej kwestii może przynieść próba identyfikacji klastrów o mniejszym rozmiarze.

Wyniki eksperymentu – podział na cztery klastry. Analizie poddano jedynie trzy klastry. Jeden z klastrów składał się z zaledwie jednego wydania projektu. Istnienie tak małego klastra nie dostarcza, z punktu widzenia predykcji międzyprojektowej, żadnych użytecznych informacji. W związku z tym klaster ten został pominięty w dalszych analizach.

Model zbudowany na podstawie klastera "1 z 4" dostarczył predykcji minimalnie dokładniejszych niż model ogólny. Jak można zobaczyć w Tabeli 5.20 był dokładniejszy o zaledwie $0,6\%$. Tak mała różnica nie okazała się być istotna statystycznie. Uzyskano prawdopodobieństwo testowe $p = 0,523$,

Tabela 5.20: Klaster "1 z 4 " – oceny modeli

	Średnia	σ
E_W	47,82	18,65
E_K	47,20	17,13

Tabela 5.21: Klaster "1 z 4" – testy hipotez

		E_W	E_K
Test	W	0,987	0,728
Shapiro–Wilka	p	0,989	0,849
Test	df	128	
Levenea	F(1,df)	0,522	
	p	0,471	
	t	0,642	
Test–t	df	64	
	p	**0,523**	

co nie pozwoliło na odrzucenie hipotezy zerowej. Nie ma zatem podstaw do stwierdzenia, że klaster "1 z 4" istnieje z punktu widzenia predykcji defektów. Do sprawdzenia hipotezy można było zastosować test t ponieważ nie zostały naruszone założenia dotyczące normalności i homogeniczności rozkładu. Wyniki testowania hipotezy przedstawiono w Tabeli 5.21.

Tabela 5.22: Klaster "2 z 4 " – oceny modeli

	Średnia	σ
E_W	64,67	15,75
E_K	43,20	7,74

Tabela 5.23: Klaster "2 z 4" – testy hipotez

		E_W	E_K
Test	W	0,906	0,464
Shapiro–Wilka	p	0,943	0,670
Test	df	6	
Levenea	F(1,df)	3,118	
	p	0,128	
	t	3,437	
Test–t	df	3	
	p	**0,041**	

Wyniki zastosowania modelu opartego o dane pochodzące z klastra "2 z 4" zaprezentowano w Tabeli 5.22. Model M_K okazał się być zdecydowanie lepszym pod względem dokładności dostarczanych predykcji. Różnica wyniosła niemal $21,5\%$. Z drugiej jednak strony klaster "2 z 4" był stosunkowo niewielki, składał się bowiem z zaledwie czterech wydań projektów. Mimo tak niewielkiego rozmiaru próbki okazało się, jak pokazano w Tabeli 5.23, że wynik ten był istotny statystycznie na poziomie istotności $\alpha = 0,05$. Są zatem podstawy do stwierdzenia, że klaster "2 z 4" istnieje z punktu widzenia modeli predykcji defektów. Do testowania hipotezy wykorzystano test t ponieważ, jak można zobaczyć w Tabeli 5.23, nie zostały naruszone jego założenia dotyczące normalności i homogeniczności rozkładu.

Normalność rozkładu sprawdzano przy pomocy testu Shapiro–Wilka, a homogeniczność przy pomocy testu Levenea.

Tabela 5.24: Klaster "3 z 4 " – oceny modeli

	Średnia	σ
E_W	51,34	17,15
E_K	52,97	18,16

Ostatni z badanych tu, a uzyskanych z podziału na cztery klastry przeprowadzonego przy pomocy metody k–średnich, klastrów to klaster "3 z 4". Dokładność predykcji modelu zbudowanego na podstawie tego klastra była o około $1,5\%$ gorsza od dokładności predykcji dostarczonych przez model ogólny. Wynik taki wskazuje na brak podstaw na uznanie istnienia klastra "3 z 4" z punktu widzenia predykcji defektów. Wniosek taki można wyciągnąć nawet bez obliczania testów statystycznych hipotez. Wyniki zastosowania modelu opartego o dane pochodzące z klastra "3 z 4" przedstawiono w Tabeli 5.24.

Udało się wykazać istnienie tylko jednego spośród zidentyfikowanych przy pomocy metody k–średnich klastrów. Dodatkowo należy jeszcze zwrócić uwagę na fakt, że rozmiar tego klastra jest stosunkowo niewielki. Natomiast z punktu widzenia predykcji międzyprojektowej zdecydowanie większą użyteczność mają duże klastry.

Wyniki eksperymentu – podział na klastry przy pomocy sieci Kohonena. W świetle wyników przedstawionych w Tabeli 5.25 trudno utrzymać tezę o istnieniu pierwszego ze zidentyfikowanych przy pomocy sieci Kohonena klastrów, czyli "przemysłowego A". Model skonstruowany na podstawie danych pochodzących z tego właśnie klastra dostarczał zdecydowanie gorszych predykcji niż model ogólny. Predykcji dokonywano na rzecz wydań projektów należących do klastra. Średnia ocena dokładności predykcji modelu skonstruowanego dla klastra jest gorsza aż o 4%. Wynik taki pozwala na stwierdzenie, że brak podstaw od uznania istnienia klastra "przemysłowego A" z punktu widzenia predykcji defektów.

Tabela 5.25: Klaster "przemysłowy A" – oceny modeli

	Średnia	σ
E_W	54,16	9,54
E_K	58,18	7,73

Tabela 5.26: Klaster "przemysłowy B" – oceny modeli

	Średnia	σ
E_W	56,35	16,59
E_K	45,05	7,81

Tabela 5.26 pokazuje dużo większą dokładność predykcji modelu zbudowanego na podstawie wydań projektów należących do klastra "przemysłowy B" niż modelu ogólnego. Model M_K dostarczył predykcji lepszych średnio o 11%. Nie może więc dziwić, że mimo niezbyt dużej liczebności klastra (składał się on z zaledwie 14 wydań projektów) wynik ten okazał się istotny statystycznie. Jak można zobaczyć w Tabeli 5.27, zastosowano test Wilcoxona i uzyskano prawdopodobieństwo testowe $p = 0,035$. Hipotezę testowano na poziomie istotności $\alpha = 0,05$. W związku z tym można odrzucić hipotezę zerową i przyjąć alternatywną. Tym samym istnieją podstawy do stwierdzenia, że klaster "przemysłowy B" istnieje z punktu widzenia predykcji defektów.

Tabela 5.27: Klaster "przemysłowy B" – testy hipotez

		E_W	E_K
Test	W	0,923	0,985
Shapiro–Wilka	p	0,243	0,995
Test	df	26	
Levenea	F(1,df)	12,778	
	p	**0,001**	
Test	t	2,103	
Wilcoxona	df	19	
	p	**0,035**	

Do testowania hipotezy zerowej zastosowano test Wilcoxona ponieważ naruszone było założenie testu t dotyczące homogeniczności rozkładu. Jak pokazuje Tabela 5.27 test Levenea dał wynik istotny statystycznie na poziomie istotności $\alpha = 0,05$. W związku z tym należało odrzucić hipotezę o homogeniczności rozkładu i zrezygnować ze stosowania testu t.

Tabela 5.28: Klaster "przemysłowo–otwarty" – oceny modeli

	Średnia	σ
E_W	55,81	22,97
E_K	50,74	20,01

Tabela 5.29: Klaster "przemysłowo–otwarty" – testy hipotez

		E_W	E_K
Test	W	0,971	0,954
Shapiro–Wilka	p	0,824	0,499
Test	df	34	
Levenea	F(1,df)	0,155	
	p	0,696	
	t	3,180	
Test–t	df	17	
	p	**0,005**	

Wyniki ewaluacji modelu skonstruowanego na podstawie projektów należących do klastra "przemysłowo–otwartego" zaprezentowano w Tabeli 5.28. Model zbudowany na podstawie tego klastra dostarczał predykcji wyraźnie dokładniejszych niż model ogólny. Różnica w przypadku średniej oceny wyniosła około 5%. Jest to zatem różnica zdecydowanie mniejsza niż w przypadku poprzednio badanego klastra "przemysłowego B". Mimo to, jak pokazuje Tabela 5.29, jest ona istotna statystycznie. Pewną rolą w uzyskaniu wyniku istotnego statystycznie odgrywa również fakt, że klaster ten składał się z większej liczby projektów, mianowicie z osiemnastu. To czy model M_K jest istotnie lepszy od modelu ogólnego M_W weryfikowano przy pomocy testu t. Uzyskano prawdopodobieństwo testowe $p = 0,005$, co pozwoliło na odrzucenie hipotezy zerowej przy poziomie istotności $\alpha = 0,05$. Innymi słowy są podstawy do tego, aby uznać, że klaster "przemysłowo–otwarty" istnieje z punktu widzenia predykcji defektów. Do testowania istnienia tego klastra można było zastosować test t ponieważ nie zostały naruszone założenia dotyczące normalności oraz homogeniczności rozkładu. Wyniki testów sprawdzających te założenia przedstawiono w Tabeli 5.29.

Tabela 5.30: Klaster "otwarty" – oceny modeli

	Średnia	σ
E_W	44,09	14,08
E_K	45,90	13,25

Wyniki analizy ostatniego ze zidentyfikowanych przy pomocy sieci Kohonena klastra przedstawiono w Tabeli 5.30. Klaster ten nazwano "otwarty". Stosowanie modelu skonstruowanego na podstawie danych pochodzących z tego klastra dostarczało predykcji mniej dokładnych niż model ogólny. Różnica wyniosła niespełna 2%. Jednak mimo tak niewielkiej różnicy, sam fakt, że model M_K okazał się być gorszy, powoduje, iż nie ma potrzeby stosowania testów statystycznych i można od razu stwierdzić brak podstaw do uznania istnienia klastera "otwartego" z punktu widzenia predykcji defektów.

Zastosowanie sieci Kohonena pozwoliło na zidentyfikowanie ciekawszych klastrów niż miło to miejsce w przypadku metodu k–średnich. Istnienie aż połowy z nich udało się statystycznie udowodnić. W przypadku metody k–średnich wykazano istnienie tylko jednego z klastrów.

Wszystkie klastry, których istnienie zostało wykazane przy pomocy testów statystycznych zostaną dokładniej opisane w podsumowaniu. Zostaną tam wymienione wszystkie wydania wchodzące w ich skład oraz zostanie przedstawiona ich charakterystyka wyrażona przy pomocy innych wielkości niż metryki oprogramowania. Charakterystyka klastrów zostanie następnie porównana z odkryciami pochodzącymi z innych badań, a dotyczącymi predykcji międzyprojektowej.

Analiza dyskryminacyjna. Analizę dyskryminacyjną przeprowadzono dla podziałów na dwa oraz na cztery klastry wykonanych przy pomocy metody k–średnich i dla klasteryzacji wykonanej przy pomocy sieci Kohonena.

W podziale na dwa klastry uzyskano lambdę Wilksa równą $0,23310$ co daje prawdopodobieństwo testowe $p < 0,0001$. Można zatem uznać z poziomem $\alpha = 0,05$ istotność uzyskanego podziału na klastry. Informacje o tym jaki był wkład poszczególnych składowych wektora korelacji w uzyskany podział przedstawiono w Tabeli 5.31. Istotność poszczególnych składowych wektora określają wartości cząstkowej lambdy Wilksa. Im mniejsze wartości tej statystyki tym większa istotność danej składowej. Informacje o wkładzie poszczególnych zmiennych można również odczytać z lambdy Wilksa. Tu należy jednak pamiętać, że statystyka ta opisuje moc dyskryminacyjną całego modelu. Wartości podane przy poszczególnych składowych informują o mocy dyskryminacyjnej modelu po wprowadzeniu do modelu danej składowej. Składowe są wprowadzane do modelu zaczynając od najistotniejszych. W związku z tym największe wartości lambdy Wilksa pojawiają się właśnie przy tych składowych, które podczas klasteryzacji odgrywały najistotniejszą rolę. Dla poszczególnych składowych wyliczono również prawdopodobieństwo testowe. Dzięki temu można ocenić wkład których składowych był istotny statystycznie. Prawdopodobieństwo testowe było mniejsze od przyjętego poziomu istotności $\alpha = 0,05$ tylko w przypadku składowej DPC. Dla składowej r(DAM) uzyskano $p = 0,050$, czyli znajdujące się dokładnie na granicy przyjętego poziomu istotności. Wynik taki oznacza, że tylko te dwie składowe odegrały istotną rolę w klasteryzacji i w związku z tym możnaby rozmiar wektora korelacji ograniczyć do tych właśnie dwóch składowych. Przedstawiona w Tabeli 5.31 tolerancja zawiera informacje o tym na ile poszczególne składowe są redundantne w stosunku do pozostałych. Tolerancja równa jeden oznacza że dana składowa w ogóle nie jest skorelowana z pozostałymi. Natomiast wartość równa zero oznacza, że dana zmienna jest całkowicie redundantna w stosunku do pozostałych. Tolerancja pokazuje zatem, że w uzyskanym podziale na klastry r(WMC), r(DIT), r(RFC), r(MFA) oraz r(CBM) wprowadzają bardzo mało informacji, które nie są dostępne w pozostałych składowych. Natomiast

Tabela 5.31: Analiza dyskryminacyjna – podział na dwa klastry

Metryka	Lambda Wilksa	Cząstkowa lambda Wilksa	F	p	Tolerancja
r(WMC)	0,236	0,988	0,816	0,369	0,067
r(DIT)	0,233	1,000	0,004	0,952	0,068
r(NOC)	0,234	0,998	0,121	0,729	0,634
r(CBO)	0,234	0,996	0,287	0,594	0,133
r(RFC)	0,236	0,986	0,955	0,332	0,083
r(LCOM)	0,234	0,998	0,137	0,712	0,116
r(Ca)	0,238	0,979	1,487	0,227	0,172
r(Ce)	0,239	0,975	1,795	0,185	0,163
r(NPM)	0,235	0,993	0,468	0,496	0,188
r(LCOM3)	0,249	0,935	4,784	0,032	0,331
r(LOC)	0,240	0,970	2,110	0,151	0,114
r(DAM)	0,247	0,946	3,974	0,050	0,295
r(MOA)	0,233	1,000	0,011	0,918	0,446
r(MFA)	0,233	1,000	0,000	0,985	0,063
r(CAM)	0,241	0,967	2,384	0,127	0,324
r(IC)	0,233	1,000	0,003	0,955	0,101
r(CBM)	0,233	0,999	0,036	0,851	0,095
r(AMC)	0,236	0,989	0,745	0,391	0,237
r(Max(CC))	0,235	0,992	0,543	0,464	0,219
r(Avg(CC))	0,233	1,000	0,002	0,962	0,179
DPC	0,297	0,786	18,782	0,000	0,438
DPDC	0,236	0,986	0,970	0,328	0,419

składowa r(NOC) jest zdecydowanie najsłabiej skorelowana z pozostałymi składowymi. Pomimo tego składowa ta nie okazała się być istotną statystycznie w procesie klasteryzacji.

Dla podziału na cztery klastry wykonanego metodą k–średnich uzyskano lambdę Wilksa równą $0,06348$. Takiej wartości lambdy odpowiada prawdopodobieństwo testowe $p < 0,0001$. Można zatem przyjąć z poziomem istotności $\alpha = 0,05$, że uzyskany podział na klastry jest istotny statystycznie. Szczegółowe informacje na temat wkładu poszczególnych składowych wektora korelacji przedstawiono w Tabeli 5.32. W przypadku tego podziału na klastry najistotniejszą rolę odegrały r(RFC), r(LCOM3), r(LOC), r(AMC) oraz DPC. Dla składowych tych otrzymywano cząstkową lambdę Wilksa mniejszą niż $0,87$, a wkład każdej z tych składowych był istotny statystycznie. Pomimo tego, po wprowadzeniu ostatniej z tych składowych otrzymano lambdę Wilksa równą $0,073$, czyli aż o $0,01$ większą od końcowej. Okazuje się więc, że zmienne nie posiadające istotnego statystycznie wkładu w sporym stopniu wpłynęły na moc dyskryminacyjną. Najmniejszy wkład w klasteryzację miały składowe r(NOC), r(LCOM), r(IC) oraz r(CBM). Dla każdej z nich cząstkowa lambda Wilksa przyjmowała wartości równe lub większe $0,995$.

Dla podział na klastry wykonanego przy pomocy sieci Kohonena uzyskano lambda Wilksa równą $0,02170$, co dało $p < 0,0001$. Można zatem przyjąć, że moc dyskryminacyjna jest istotna statystycznie na poziomie $\alpha = 0,05$. Szczegóły na temat poszczególnych składowych wektora korelacji, a właściwie to ich wkładu w przeprowadzoną klasteryzację zaprezentowano w Tabeli 5.33. Zgodnie z przedstawionymi tam danymi największą rolę odegrały składowe: r(WMC), r(CBO), r(Ca), r(Ce) oraz r(MOA). Największa wartość cząstkowej lambdy Wilksa dla tych składowych to $0,703$. Podobnie jak w przypadku podziału na cztery klastry metodą k–średnich, wprowadzenie do modelu tylko tych

Tabela 5.32: Analiza dyskryminacyjna – podział na cztery klastry

Metryka	Lambda Wilksa	Cząstkowa lambda Wilksa	F	p	Tolerancja
r(WMC)	0,064	0,988	0,404	0,669	0,069
r(DIT)	0,065	0,983	0,583	0,561	0,072
r(NOC)	0,064	0,996	0,133	0,875	0,610
r(CBO)	0,066	0,955	1,582	0,213	0,134
r(RFC)	0,078	0,809	7,911	0,001	0,080
r(LCOM)	0,064	0,995	0,166	0,847	0,101
r(Ca)	0,066	0,966	1,172	0,316	0,187
r(Ce)	0,065	0,976	0,827	0,442	0,152
r(NPM)	0,066	0,958	1,461	0,239	0,183
r(LCOM3)	0,081	0,788	9,014	0,000	0,340
r(LOC)	0,088	0,723	12,849	0,000	0,088
r(DAM)	0,071	0,892	4,056	0,022	0,348
r(MOA)	0,066	0,967	1,141	0,326	0,466
r(MFA)	0,065	0,981	0,658	0,521	0,084
r(CAM)	0,069	0,915	3,107	0,051	0,491
r(IC)	0,064	0,995	0,170	0,844	0,093
r(CBM)	0,064	0,999	0,026	0,974	0,086
r(AMC)	0,073	0,869	5,058	0,009	0,222
r(Max(CC))	0,067	0,947	1,875	0,161	0,230
r(Avg(CC))	0,069	0,924	2,764	0,070	0,184
DPC	0,120	0,529	29,819	0,000	0,403
DPDC	0,068	0,937	2,252	0,113	0,400

najistotniejszych składowych, zaowocowało dużo niższą mocą dyskryminacyjną niż ta, którą można osiągnąć po wprowadzeniu wszystkich składowych. Po wprowadzeniu najistotniejszych składowych uzyskano lambdę Wilksa równą $0,031$, a ostateczna lambda Wilksa wyniosła $0,022$. Najmniej istotną składową okazała się być DPC, cząstkowa lambda Wilksa wyniosła w jej przypadku aż $0,951$. Jest to bardzo ciekawe spostrzeżenie z uwagi na fakt, że ta właśnie składowa odgrywała najistotniejszą rolę w podziałach wykonywanych metodą k–średnich.

5.3 Zagrożenia dla ważności uzyskanych wyników

Eksperymenty dotyczące predykcji międzyprojektowej od strony struktury eksperymentu mają wiele cech wspólnych z eksperymentami identyfikującymi czynniki poprawiające dokładność predykcji. W szczególności w obu typach eksperymentów wykorzystywano te same dane. W związku z powyższym większość zagrożeń dla ważności uzyskanych wyników jest identyczna z tymi przedstawionymi w Rozdziale 4.3.

5.3.1 Walidacja wewnętrzna

Podobnie jak w przypadku eksperymentu dotyczącego czynników poprawiających dokładność predykcji nie zidentyfikowano tu poważniejszych zagrożeń dla walidacji wewnętrznej. W eksperymentach omawianych w tym rozdziale również nie brano pod uwagę metryk istotności defektów oraz nie naruszono żadnego z trzech, omówionych w Rozdziale 4.3 warunków walidacji wewnętrznej.

Tabela 5.33: Analiza dyskryminacyjna – sieć Kohonena

Metryka	Lambda Wilksa	Cząstkowa lambda Wilksa	F	p	Tolerancja
r(WMC)	0,035	0,615	6,265	0,002	0,022
r(DIT)	0,024	0,917	0,904	0,451	0,144
r(NOC)	0,026	0,825	2,119	0,119	0,574
r(CBO)	0,038	0,568	7,603	0,001	0,051
r(RFC)	0,026	0,831	2,036	0,130	0,024
r(LCOM)	0,024	0,922	0,849	0,478	0,073
r(Ca)	0,038	0,575	7,384	0,001	0,061
r(Ce)	0,031	0,703	4,220	0,013	0,039
r(NPM)	0,025	0,866	1,553	0,221	0,036
r(LCOM3)	0,026	0,842	1,880	0,154	0,137
r(LOC)	0,024	0,893	1,195	0,329	0,113
r(DAM)	0,026	0,849	1,780	0,172	0,239
r(MOA)	0,044	0,494	10,234	0,000	0,172
r(MFA)	0,026	0,839	1,917	0,148	0,127
r(CAM)	0,028	0,776	2,890	0,052	0,221
r(IC)	0,027	0,794	2,601	0,070	0,065
r(CBM)	0,025	0,881	1,345	0,278	0,088
r(AMC)	0,027	0,813	2,303	0,097	0,169
r(Max(CC))	0,025	0,881	1,346	0,278	0,077
r(Avg(CC))	0,024	0,922	0,843	0,481	0,109
DPC	0,023	0,951	0,519	0,673	0,124
DPDC	0,025	0,852	1,740	0,180	0,103

5.3.2 Walidacja zewnętrzna

W omawianym w tym rozdziale eksperymencie badano bardzo liczny zbiór różnorodnych projektów. Jednak z uwagi na mnogość technologii oraz metod wytwarzania systemów informatycznych nie można przyjąć, że zbiór ten dobrze reprezentuje wszystkie występujące w populacji cechy. Efektem klasteryzacji jest pogrupowanie projektów w zbiory, których elementy cechują się podobną charakterystyką. Absolutnie nie można przyjąć, że podział taki wyczerpuje wszystkie możliwe do zidentyfikowania klastry projektów. Z pewnością istnieje wiele klastrów, których w omawianym tu eksperymencie zidentyfikować nie było można z uwagi na zbyt małą liczbę badanych projektów.

W podsumowaniu do tego rozdziału podano charakterystykę zidentyfikowanych klastrów. Wyszukano cechy wspólne dla wszystkich projektów, które zostały do danego klastra zakwalifikowane. Można mieć wątpliwości zarówno co do tego, czy żadna z istotnych cech nie została przeoczona, jak i co do tego, czy któraś ze wskazanych cech nie jest cechą przypadkową, nie mającą wiele wspólnego z rozstrzyganiem o przynależności do danego klastra. Zweryfikowanie słuszności charakterystyk zidentyfikowanych klastrów wymaga przeprowadzenia eksperymentu weryfikującego opartego o dane pochodzące z innych projektów. Dokonano jedynie częściowej walidacji charakterystyk przez porównanie z wynikami uzyskiwanymi przez innych badaczy [119]. Na podstawie tego porównania można stwierdzić, że nie uzyskano wyników sprzecznych. Niemniej nie wyczerpuje to potrzeby przeprowadzenia eksperymentu weryfikującego.

5.3.3 Walidacja konkluzji

Wyniki eksperymentu są efektem testowania hipotez statystycznych na poziomie istotności $\alpha = 0,05$. Poprawność podziału na klastry zweryfikowano dodatkowo przy pomocy analizy dyskryminacyjnej. Do obliczeń statystycznych stosowano pakiet Statistica i posługiwano się procedurami zaproponowanymi w [99], [100] i [101].

5.3.4 Walidacja konstrukcji

Na etapie przygotowywania danych do eksperymentu trzeba było rozwiązać problem odtworzenia powiązania pomiędzy defektami a klasami. Do tego celu wykorzystywano informacje pochodzące z systemów śledzenia defektów oraz z systemów kontroli wersji. Jak jednak powszechnie wiadomo [26], [35], [112], [120] podejście takie wiąże się z pewnymi problemami. Najpierw należy w systemie śledzenia defektów odróżnić te zgłoszenia, które reprezentują defekty, od tych które są powiązane z opracowywaniem nowych funkcjonalności, co czasami jest trudnym zadaniem [112] i może stać się źródłem pomyłek. Następnie defekty te są łączone z klasami na podstawie komentarzy z systemu kontroli wersji. Wykonanie tego manualnie dla dużych projektów nie jest możliwe, natomiast wykonanie tego automatycznie wiąże się z możliwością wystąpienia błędów [35]. Tu posłużono się metodą zautomatyzowaną, mianowicie wykorzystano specjalnie do tego celu opracowany program *BugInfo*.

Reguły określające co należy interpretować jako defekt mogą się różnić pomiędzy projektami. Antoniol i in. [1] pokazali, analizując projekty *Eclipse*, *JBoss* oraz *Mozilla*, że znaczna część zgłoszeń z systemu śledzenia defektów nie reprezentuje defektów. Zbierając dane przeprowadzono automatyczną eliminacji zgłoszeń, które były w systemie śledzenia defektów zaklasyfikowane do jednej z kategorii używanych zazwyczaj do obsługi zgłoszeń nie związanych z defektami. Działanie takie nie gwarantuje niestety wyeliminowania wszystkich zgłoszeń nie będących defektami.

Istotny problem sprawiają również klasy anonimowe i wewnętrzne. Systemy kontroli wersji (SubVersion, CVS) działają na poziomie pliku. Nie sposób zatem rozstrzygnąć czy defekt odnotowany w logach systemu kontroli wersji pochodzi z klasy wewnętrznej, czy zewnętrznej. Problemy te spowodowały, że zaleca się ignorowanie tych klas [1], [16], [17]. Istotność tego problemu jest bardzo łatwa do oszacowania, ponieważ liczba klas anonimowych i wewnętrznych jest możliwa do wyliczenia. W danych wykorzystanych do opisanego w tym rozdziale eksperymentu klasy te stanowiły $9,04\%$ wszystkich klas. Odsetek klas anonimowych i wewnętrznych jest w przypadku tego eksperymentu większy niż w eksperymencie z Rozdziału 4, ponieważ w tym eksperymencie wykorzystywano dodatkowo projektu studenckie, których zawierają stosunkowo dużo takich klas.

Podobne problemy sprawiają takie operacje jak zmiana nazwy klasy, albo przeniesienie jej pomiędzy pakietami. Operacje te powodują przerwanie historii klasy i tym samym uniemożliwiają zidentyfikowania powiązania pomiędzy stanem klasy z przed i po operacji. Po takiej operacji klasa jest interpretowana jako nowa, co ma negatywny wpływ zarówno na poprawność identyfikowania powiązań z defektami jak i na niektóre metryki procesu.

Defekty były przypisywane do zgłoszeń na podstawie daty naprawienia zgłoszenia, gdyż niezbędny do zidentyfikowania defektu komentarz z systemu śledzenia defektów jest wprowadzany dopiero w chwili zatwierdzania zmian rozwiązujących problem. Zdecydowanie ciekawszy byłby eksperyment, w którym udałoby się defekty łączyć z tymi wydaniami projektów, w których defekty te były wprowadzane.

Stosowano tylko jedną metodę konstrukcji modeli predykcji defektów, mianowicie postępującą regresję liniową. W związku z tym trudno zawyrokować na ile wyniki tu uzyskane mają zastosowanie do eksperymentów, w których stosuje się inne metody konstrukcji modeli. Z drugiej jednak strony zastosowana metoda jest najmniej skomplikowana pod względem struktury uzyskiwanego modelu.

Jeżeli więc możliwe jest generalizowanie wyników uzyskanych dla jednej metody na pozostałe, to zdecydowanie większe szanse na powodzenie ma generalizacja wychodząca od metod prostych (takich jak tu zastosowana postępująca regresja liniowa) do skomplikowanych (np. sieci neuronowe, regresja logistyczna). Wydaje się zatem, że dobór metody konstrukcji modelu predykcji defektów był optymalny pod kątem możliwości uogólniania uzyskanych wyników. Niemniej rozstrzygnięcie kwestii możliwości generalizowania wyników na inne metody wymaga replikacji eksperymentu z wykorzystaniem innej metody konstruowania modelu predykcji defektów.

5.4 Podsumowanie

W ramach badań dotyczących predykcji międzyprojektowej poszukiwano takich klastrów złożonych z projektów programistycznych, żeby w ramach danego klastra można było stosować jeden i ten sam model predykcji defektów do wszystkich projektów należących do tego klastra. W tym celu przeprowadzono dwa typy eksperymentów do których wykorzystano 92 wydania 38 projektów. W pierwszym założono a priori istnienie trzech klastrów, mianowicie klastra projektów przemysłowych, projektów otwartych oraz projektów studenckich. W drugim typie eksperymentów wykorzystano metody eksploracji danych do zidentyfikowania klastrów projektów programistycznych. W obu typach eksperymentów badano istnienie zidentyfikowanych wcześniej klastrów. W tym celu konstruowano modele predykcji defektów i oceniano dokładność uzyskanych przy ich pomocy predykcji. Uzyskiwane wyniki oceniano przy pomocy testów statystycznych. Tym sposobem udało się wykazać istnienie trzech klastrów. Adekwatność uzyskanych podziałów na klastry oceniono przy pomocy analizy dyskryminacyjnej. We wszystkich przypadkach uzyskano statystycznie istotną moc dyskryminacyjną. Dodatkowo przy pomocy analizy dyskryminacyjnej wskazano czynniki, które decydowały o kształcie poszczególnych podziałów.

5.4.1 Uzyskane wyniki

Tabela 5.34: Podsumowanie wyników eksperymentu wstępnego

Modele przemysłowe w projektach przemysłowych		Modele otwarte w projektach otwartych		Modele studenckie w projektach studenckich	
Nieznacznie lepsze od nieprzemysłowych	—	Nieznacznie lepsze od nieotwartych	—	Nieznacznie lepsze od niestudenckich	—
Nieznacznie lepsze od otwartych	—	Istotnie lepsze od przemysłowych	↗	Gorsze od od otwartych	↘
Istotnie lepsze od studenckich	↗	Istotnie lepsze od studenckich	↗	Nieznacznie lepsze od przemysłowych	—

'↗' – lepszy, wynik istotny statystycznie;
'↘' – gorszy;
'–' – lepszy, wynik nie istotny statystycznie;

Podsumowanie eksperymentu wstępnego zaprezentowano w Tabeli 5.34. Można z pewnym uproszczeniem przyjąć, że udało się częściowo wykazać istnienie klastra projektów otwartych. Natomiast w przypadku klastra projektów studenckich uzyskano zaskakująco słabe wyniki. Wskazywać to może na

sporą dozę chaosu w tych projektach i stawia pod znakiem zapytania sens stosowania takich projektów w badaniach empirycznych dotyczących modeli predykcji defektów.

Tabela 5.35: Podsumowanie wyników eksperymentu z eksploracją danych

Klaster	Ocena względem modelu referencyjnego	Wynik
1 z 2	0,06	nieistotny stat.
2 z 2	-0,23	–
1 z 4	0,61	nieistotny stat.
2 z 4	21,47	istotny stat.
3 z 4	-1,63	–
4 z 4	Tylko jedno wydanie w klastrze	
Przemysłowy A	-4,02	–
Przemysłowy B	6,03	istotny stat.
Przemysłowo otwarty	5,07	istotny stat.
Otwarty	-1,81	–

Podsumowanie eksperymentu wykorzystującego eksplorację danych zaprezentowano w Tabeli 5.35. Łącznie badano istnienie 10 klastrów. 4 z nich zostały zidentyfikowane przy pomocy sieci Kohonena, a 6 przy pomocy k–średnich. Jeden klaster spośród tych 6 był zdegenerowany. Składał się tylko z jednego wydania, jednego projektu. W związku z tym klaster ten pominięto w analizach. Przeprowadzone analizy wykazały z poziomem istotności $\alpha = 0,05$ istnienie trzech klastrów. Są to "2 z 4", "Przemysłowy B" oraz "Przemysłowo otwarty"

5.4.2 Charakterystyki zidentyfikowanych klastrów

Klaster "2 z 4" to bardzo mały klaster, ale mimo to bardzo różnorodny pod względme zaklasyfikowanych do niego wydań projektów. W jego skład wchodzą *Apache Ant* w wersji 1.4 i trzy projekty studenckie dotyczące zagadnień z trzech różnych dziedzin: program na telefony komórkowe, wykonana w *EJB* aplikacja internetowa wspierająca pracę wydawców gazety studenckiej oraz opierający się na platformie *Spring* sklep internetowy. Członkowie tego klastra są tak różnorodni, że nie udało się dla nich znaleźć spójnej charakterystyki. Klaster ten w aktualnej postaci nie jest więc nazbyt użyteczny.

Klaster "Przemysłowy B" składa się około połowy projektów przemysłowych. Wszystkie zakwalifikowane do tego klastra projekty to rozwiązania przygotowywane na specjalne zamówienie klienta. Wszystkie tego typu projekty przemysłowe (nie tylko te które należą do klastra "Przemysłowy B") rozwijane były w sformalizowanym, sterowanym planem procesie wytwarzania oprogramowania. Wszystkie one zostały już z sukcesem zainstalowane w środowisku klienta. Kolejne wydania tych projektów składają się z poprawek naprawiających błędy oraz z implementacji nowych funkcjonalności wynikających bądź to z życzeń klienta, bądź też ze zmian w prawie. Wszystkie te projekty należą do branży ubezpieczeniowej. Nie wszystkie projekty przemysłowe pasujące do powyższego opisu zostały zakwalifikowane do klastra "Przemysłowy B". W związku z tym przeprowadzono ich dokładniejszą analizę opierającą się na wywiadach przeprowadzonych z osobami realizującymi te projektu. Okazało się, że pomiędzy tymi projektami przemysłowymi, które zostały zakwalifikowane do klastra "Przemysłowy B" a pozostałymi, istniały istotne różnice w fazie testów. Prawie wszystkie wydania projektów zakwalifikowane do klastra "Przemysłowy B" były testowane ręcznie. Natomiast w wydaniach, które do tego klastra nie zostały zakwalifikowane powszechnie stosowano testy automatyczne na poziomie regresyjnych testów funkcjonalnych. Czynnik związany ze stosowaną metodą testowania wyjaśnił przynależność wszystkich za wyjątkiem trzech wydań projektów. Istniały mianowicie trzy

wydania projektów, które były testowane ręcznie, a mimo to nie zostały zakwalifikowane do klastra "Przemysłowy B". Po dalszej analizie okazało się, że te trzy wydania były rozwijane przez krótszy okres czasu niż pozostałe. We wszystkich badanych projektach przemysłowych stosowano praktykę polegającą na przygotowywaniu dwóch wydań rozwijanego systemu w ciągu roku. Powoduje to, że prace nad każdym wydaniem trwają około sześć miesięcy. Istnieje jednak kilka wydań, które się od tej zasady wyłamały i były opracowywane przez krótszy okres czasu. W szczególności od tej zasady wyłamały się te trzy wydania, które nie zostały zakwalifikowane do klastra "Przemysłowy B".

Klaster "Przemysłowo otwarty" składa się z *Apache Forrest* w wersjach 0.7 oraz 0.8, *Apache POI* w wersjach 2.5.1 oraz 3.0, *Apache Xalan* w wersjach 2.4, 2.5, 2.6 oraz 2.7, *Apache Xerces* w wersjach 1.1.0, 1.2.0, 1.3.0 oraz 1.4.4, *JEdit* w wersjach 3.2.1, 4.1, 4.2 oraz 4.3 i dwóch wersji tego projektu przemysłowego, który jest standardowym narzędziem wspierającym zapewnianie jakości w projekcie programistycznym. Klaster ten składa się zatem z kilku różnych projektów, które były rozwijane przez różne firmy w różnych procesach wytwarzania oprogramowania. Dziedzina projektów również nie jest jednolita, ale mimo to można w tej kwestii znaleźć pewne cechy wspólne. Wszystkie należące do tego klastra projekty mają wiele wspólnego z przetwarzaniem tekstu. Wszystkie, za wyjątkiem *JEdit*, wykorzystują systemy *Jira* lub *Bugzilla* jako system śledzenia defektów. Wszystkie, również za wyjątkiem *JEdit*, są rozwijane przez międzynarodowe zespoły średniej wielkości. Największy zespół składał się z 25 osób. W większości przypadków było to jednak dokładnie 11 osób. We wszystkich należących do tego klastra projektach wykorzystywano *SubVersion* do zarządzania wersjami kodu. Istnieją również podobieństwa dotyczące procesu testowania. Powszechnie stosowano w tych projektach wysoki poziom automatyzacji w testach.

Podsumowanie charakterystyk klastrów przedstawiono w Tabeli 5.36.

Tabela 5.36: Charakterystyki klastrów

Klaster	Cechy wspólne
2 z 4	–
Przemysłowy B	indywidualne zamówienie klienta, sformalizowany i sterowany planem proces, już zainstalowane u klienta, branża ubezpieczeniowa, testy manualne, taki sam czas prac nad wydaniem, wykorzystywana baza danych, przemysłowe – wytworzone przez tą samą firmę
Przemysłowo otwarty	przetwarzanie tekstu, wykorzystywany *SubVersion* oraz *Jira* lub *Bugzilla*, średniego rozmiaru międzynarodowy zespół, automatyzacja w procesie testowania, niewykorzystywana baza danych

5.4.3 Porównanie z wynikami innych badań

Badania dotyczące predykcji międzyprojektowej prowadzili między innymi Zimmerman i inni [119]. W swoim artykule opracowali oni drzewo decyzyjne pozwalające ocenić szansę powodzenia predykcji międzyprojektowej. Jeżeli wyniki badań przedstawionych w tym rozdziale są spójne z wynikami Zimmermana, to wśród cech charakterystycznych klastrów powinny pojawić się takie, które we wspomnianym drzewie zwiększają szansę na powodzenie predykcji międzyprojektowej, a nie powinny pojawić się takie, które zmniejszają szansę udanej predykcji. Czynniki wskazane w [119] będą konfrontowane

tylko z klastrami "Przemysłowy B" oraz "Przemysłowo otwarty". Nie udało się zidentyfikować cech charakterystycznych klastra "2 z 4", więc nie będzie on w dalszych rozważaniach brany pod uwagę.

Zaproponowane przez Zimmermana drzewo decyzyjne wskazuje, że szansa na skuteczną predykcję międzyprojektową jest większa, jeżeli model budowany jest na podstawie projektu składającego się z większej liczby klas, a stosowany w projektach z mniejszą liczbą klas. Jest to relacja niesymetryczna i w związku z tym nie przystaje ona do koncepcji klastrów. Niemniej można ją skonfrontować z liczbą klas w poszczególnych projektach należących do analizowanych klastrów. Liczba klas w klastrze "Przemysłowy B" waha się do 2286 do 4622. Występuje tam również jedno odstające wydanie, składające się z zaledwie 1694 klas. Wydania projektów należące do klastra "Przemysłowo otwarty" składają się z od 162 do 909 klas. Występują dwa wydania odstające pod względem rozmiaru, składające się z 29 oraz 32 klas.

Drugi czynnik zwiększający szansę predykcji międzyprojektowej zidentyfikowany w [119] to fakt używania bazy danych. Jeżeli oba projekty nie wykorzystują bazy danych, to powinno to ułatwiać skuteczne przeprowadzenie predykcji pomiędzy tymi projektami. Wszystkie wydania projektów wchodzące w skład klastra "Przemysłowy B" wykorzystują bazę danych, natomiast spośród wchodzących w skład klastra "Przemysłowo otwarty" żadne z bazy danych nie korzysta.

Trzecim czynnikiem zidentyfikowanym w [119] jest firma odpowiedzialna za opracowywanie danego projektu. Zaprezentowano listę konkretnych firm, które zdaniem autorów miały pozytywny, lub też negatywny wpływ na dokładność predykcji międzyprojektowej. W badanych w tym rozdziale projektach wystąpił tylko jeden ze wskazanych w [119] wytwórców oprogramowania, mianowicie *Apache*. Zdaniem Zimmermana i innych [119] fakt, że dwa projekty były wytwarzane przez *Apache*, powoduje zwiększenie dokładności predykcji międzyprojektowej. Z drugiej strony stwierdzono w [119], że predykcja pomiędzy projektami wytworzonymi przez różne firmy zmniejsz dokładność predykcji międzyprojektowej. Stwierdzenia dotyczące czynnika – wytwórca oprogramowania znalazły tylko częściowe potwierdzenie w uzyskanym podziale na klastry. Wszystkie projekty, które weszły w skład klastra "Przemysłowy B" były rozwijane przez *Capgemini–sd&m*. Projekty wchodzące w skład klastra "Przemysłowo otwarty" były wytwarzane przez trzy firmy. Zdecydowana większość (12 wydań) zostało opracowanych przez *Apache*, 4 wydania opracowała społeczność *JEdit*, a 2 pochodzą z *Capgemini–sd&m*. Klaster "Przemysłowy B" nie łamie zatem zasady dotyczące predykcji pomiędzy projektami wytworzonymi przez różne firmy i nie ma nic wspólnego z zasadą dotyczącą produktów *Apache*. Klaster "Przemysłowo otwarty" częściowo łamie zasadę dotyczące predykcji pomiędzy projektami wytworzonymi przez różne firmy, ale za to częściowo wspiera zasadę dotyczącą produktów *Apache*.

Rozdział 6

Praktyczne zastosowanie uzyskanych wyników

Tematem tego rozdziału jest zrealizowane w przemyśle wdrożenie modelu predykcji defektów. Wdrożenie to zostało oryginalnie opisane w raporcie technicznym [70] przygotowanym w firmie *Capgemini Polska*.

6.1 Motywacja i kontekst wdrożenia

Potrzeba wdrożenia narzędzia umożliwiającego dokonywanie predykcji defektów zrodziła się w drodze zaistnienia specyficznych warunków projektowych. W projekcie, o którym mowa, pojawiło się nadspodziewanie dużo żądań nowej funkcjonalności ze strony klienta. W efekcie wystąpił niedobór zasobów ludzkich niezbędnych do zrealizowania powstałych zamówień. Zdecydowano ograniczyć zasoby przeznaczane na utrzymywanie i konserwowanie starej funkcjonalności, a skoncentrować się na nowych zamówieniach. Utrzymywanie i konserwowanie starej funkcjonalności w głównej mierze sprowadzało się do naprawiania wykrytych defektów. W szczególności stały się odczuwalne niedobory w zespole testerów. W stosowanym procesie wytwarzania oprogramowania zakładano, że każda modyfikacja powinna wiązać się z wyspecyfikowaniem, przeprowadzeniem oraz udokumentowaniem sprawdzających jej poprawność testów. W zaistniałych warunkach zdecydowano, że z uwagi na ogrom prac związanych z testowaniem nowej funkcjonalności, zostaną ograniczone prace dotyczące testowania modyfikacji starej funkcjonalność, czyli testy sprawdzające poprawność usunięcia defektu z systemu. Testy przynoszą realne korzyści wtedy, kiedy prowadzą do wykrycia błędów popełnionych na którejś z wcześniejszych faz procesu wytwarzania oprogramowania. Jeżeli istniałaby możliwość przewidzenia, które modyfikacje wprowadzają błędy, to można by ograniczyć testy tylko do tych właśnie modyfikacji. Przewidywanie błędów jest natomiast podstawową rolą modeli predykcji defektów. Naturalnie zatem pojawiła się przesłanka do ich zastosowania.

Projekt, którego dotyczą niniejsze rozważania, pochodzi z branży ubezpieczeniowej. Jest rozwijany od kilku lat i doczekał się już udanego wdrożenia po stronie klienta. Projekt ten jest rozwijany przez międzynarodowy zespół programistów, testerów i analityków. Łącznie pracuje nad nim około 30 osób podzielonych na trzy zespoły. Istnieją dwa zespoły złożone z programistów oraz analityków biznesowych odpowiedzialne za specyfikację, projektowanie oraz implementację i zespół testerów odpowiedzialny za testy. Każdy z zespołów liczy około 10 osób. Projekt ten to kompleksowy, monoli-

tyczny system o architekturze klient – serwer zrealizowany w języku Java z wykorzystaniem systemu bazodanowego.

6.2 Zastosowany model predykcji defektów

Jak wynika z zarysowanego powyżej kontekstu wdrożenia, potrzebny jest nieco inny model niż te, które badano w poprzednich rozdziałach. W niniejszej rozprawie analizowano modele dokonujące predykcji dla klas. W projekcie w którym miało miejsce wdrożenie zaistniała natomiast potrzeba stosowania modelu predykcji defektów dokonującego predykcji dla zgłoszenia w systemie śledzenia defektów. Innymi słowy potrzebny był model, który będzie w stanie przewidywać, które defekty zostały poprawnie naprawione, a które nie. Popełnienie błędu w procesie naprawiania defektu powinno zakończyć się wykryciem tego błędu przez testerów i przekazaniem tego defektu do ponownej analizy i naprawy.

Model predykcji defektów miał zatem za zadanie przeanalizowanie historii defektu do momentu rozpoczęcia fazy testów i stwierdzenie na tej podstawie, czy defekt został poprawnie naprawiony. Jeżeli model wskazałby, że istnieje duża szansa na to, że programista popełnił błąd naprawiając defekt, to defekt taki zostałby gruntownie przetestowany. W przeciwnym razie poprawka zostałaby dołączona do rozwijanego systemu bez testów.

Wdrażany model powinien w związku z tym estymować inną zmienną. Analizowane w niniejszej rozprawie modele dokonywały predykcji liczby defektów. Wdrażany model powinien estymować zmienną binarną określającą czy dany defekt został poprawnie naprawiony. W raporcie [70] zmienną tą, posługując się terminologią systemu śledzenia defektów, nazwano *Reopen*. Zmienna *Reopen* równa wartości *tak* oznacza, że defekt został błędnie naprawiony i w związku z tym należy go przetestować. Wartość równa nie *nie* reprezentuje natomiast poprawną naprawę.

6.2.1 Metryki

Z uwagi na zmianę artefaktu, którego dotyczyć ma predykcja zaproponowano również nieco inny zestaw metryk, zmiennych niezależnych w modelu:

- *CRNumber* – metryka o wartościach nominalnych wskazująca w jakiej funkcjonalności znaleziono defekt. Wyróżniano tu następujące wartości: nowa funkcjonalność wprowadzona na życzenie klienta, nowa funkcjonalność wprowadzona w związku ze zmianą prawa oraz stara funkcjonalność.
- *Keywords* – metryka składająca się z listy słów kluczowych stosowanych w systemie śledzenia defektów, a przypisanych danemu defektowi.
- *Priority* – priorytet defektu.
- *SollAufwand* – oszacowanie nakładu pracy potrzebnego na naprawienie defektu.
- *IstAufwand* – rzeczywisty nakład pracy poniesiony na naprawienie defektu.
- *Product* – podsystem w którym znajdował się defekt.
- *Component* – moduł w którym znajdował się defekt. Moduł jest częścią składową podsystemu.
- *NrOfAssignedTo* – liczba różnych osób, które dla danego defektu pełniły rolę *AssignedTo*. Metryka powiązana z badaną metryką NDC.

- *NrOfStatuses* – metryka pokazująca ile razy był zmieniany status defektu zanim został naprawiony.
- *NrOfTargetMilestones* – metryka pokazująca ile razy zmieniany był kamień milowy reprezentujący termin, do którego defekt powinien zostać naprawiony.
- *NrOfAttachements* – liczba załączników do defektu.
- *NrOfCommentsBeforeFirstTest* – liczba komentarzy wstawionych przed rozpoczęciem testów defektu.
- *TimeToFirstAssignedTo* – czas jaki upłynął od zgłoszenia defektu do przydzielenia go programiście do naprawy, czyli do pierwszej zmiany statusu defektu na *AssignedTo*.
- *AllRevisions(Java)* – suma wszystkich wersji powstałych w ramach prac nad danym defektem, wszystkich modyfikowanych w ramach prac nad tym defektem plików z kodem źródłowym. Metryka ta jest blisko związana z metryką NR.
- *MaxRevisions(Java)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki z kodem źródłowym, które były modyfikowane w ramach naprawiania danego defektu. Wartość metryki będzie wtedy równa liczbie modyfikacji wykonanych w ramach naprawiania danego defektu w tym pliku, który był modyfikowany największą liczbę razy. Metryka ta jest zbliżona do metryki NR.
- *MeanAuthor(Java)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki z kodem źródłowym, które były modyfikowane w ramach naprawiania danego defektu. Wartość metryki reprezentuje wtedy średnią liczbę różnych autorów modyfikujących jeden z tych plików. Metryka ta jest zbliżona do metryki NDC.
- *MaxAuthor(Java)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki z kodem źródłowym, które były modyfikowane w ramach naprawiania danego defektu. Wartość tej metryki jest wtedy równa liczbie różnych autorów tego z tych plików, który był modyfikowany przez największą liczbę różnych autorów. Metryka ta jest zbliżona do metryki NDC.
- *AllRevisions(XML)* – suma wszystkich wersji powstałych w ramach prac nad danym defektem, wszystkich modyfikowanych w ramach prac nad tym defektem plików typu XML. Metryka ta jest blisko związana z metryką NR.
- *MaxRevisions(XML)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki typu XML, które były modyfikowane w ramach naprawiania danego defektu. Wartość metryki będzie wtedy równa liczbie modyfikacji wykonanych w ramach naprawiania danego defektu w tym pliku, który był modyfikowany największą liczbę razy. Metryka ta jest zbliżona do metryki NR.
- *MeanAuthor(XML)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki typu XML, które były modyfikowane w ramach naprawiania danego defektu. Wartość metryki reprezentuje wtedy średnią liczbę różnych autorów modyfikujących jeden z tych plików. Metryka ta jest zbliżona do metryki NDC.
- *MaxAuthor(XML)* – aby wyliczyć tą metrykę należy wziąć pod uwagę wszystkie pliki typu XML, które były modyfikowane w ramach naprawiania danego defektu. Wartość tej metryki jest wtedy równa liczbie różnych autorów tego z tych plików, który był modyfikowany przez największą liczbę różnych autorów. Metryka ta jest zbliżona do metryki NDC

- *NrOfPropertyFiles* – liczba zmodyfikowanych w ramach naprawiania danego defektu plików typu Properties.
- *NrOfSQLFiles* – liczba zmodyfikowanych w ramach naprawiania danego defektu plików typu SQL.
- *NrOfXMLFiles* – liczba zmodyfikowanych w ramach naprawiania danego defektu plików typu XML.
- *NrOfJavaFiles* – liczba zmodyfikowanych w ramach naprawiania danego defektu plików typu Java.

Przedstawione tu nazwy metryk są spójne z nazwami wykorzystanymi w raporcie [70]. Oprócz powyższych, specyficznych dla zgłoszenia w systemie śledzenia defektów metryk wykorzystano wszystkie badane w przedstawionych w niniejszej rozprawie metryki produktu. Z uwagi na fakt, że w ramach prac nad defektem może zostać zmodyfikowanych więcej niż jedna klasa, dla każdej metryki produktu dla danego defektu można otrzymać zbiór wartości. Dla każdej zmodyfikowanej klasy jedna wartość danej metryki produktu. Aby umożliwić zastosowanie metryk produktu w modelu predykcji defektów należało przeskalować opisany powyżej zbiór przeskalować na wartość, lub wektor wartości, którego rozmiar nie zależałby od liczby zmodyfikowanych klas. W tym celu dla każdej metryki produktu wyliczono jej wartość maksymalną oraz średnią w zbiorze zmodyfikowanych klas. Podczas budowy modelu predykcji defektów wykorzystano, obok wymienionych powyżej metryk specyficznych dla zgłoszenia w systemie śledzenia defektów, wykorzystano wartości średnie i maksymalne.

6.2.2 Konstrukcja modelu

Zanim przystąpiono do właściwej budowy modelu przeprowadzono wstępną analizę danych, w ramach której oczyszczono zebrane dane. Czyszczenie danych polegało w głównej mierze na rozwiązaniu problemu brakujących wartości niektórych metryk. W przypadku zmiennych nominalnych wprowadzono w tym celu nową wartość *EMPTY*, którą zastąpiono brakujące dane. Wśród analizowanych zgłoszeń z systemu śledzenia defektów występowały również takie, których naprawa nie wiązała się z modyfikowaniem żadnej klasy Javy. W takiej sytuacji brakowało danych niezbędnych do wyliczenia metryk produktu i w efekcie powstawały braki danych. Braki te uzupełniono wartościami średnimi przyjmowanymi przez poszczególne metryki produktu w badanym projekcie.

Wśród zgromadzonych danych pojawiły się zmienne mierzone na skali nominalnej. Zmienne tego typu niezbyt dobrze sprawdzają się w regresji liniowej. Z drugiej strony jednym z wymagań projektowych była czytelność modelu. Osobom, które tym modele miały się posługiwać zależało na tym, aby przesłanki na podstawie których model wyciąga wnioski, były dla nich jasne. Wymóg ten doskonale spełniają drzewa decyzyjne. Drzewa decyzyjne dodatkowo świetnie sobie radzą ze zmiennymi mierzonymi na skali nominalnej.

Skonstruowano dwa różne modele, każdy z nich oparty na drzewie decyzyjnym innego typu, a następnie wybrano ten model, który dostarczał dokładniejszych predykcji. Wykorzystano mianowicie drzewo C&RT oraz CHAID.

Metoda C&RT jest oparta na rekursywnym partycjonowaniu. Metoda ta umożliwia zarówno uzyskanie drzewa regresyjnego jak i drzewa klasyfikującego, co jest źródłem akronimu zastosowanego w nazwie. Drzewo C&RT jest ciągiem decyzji binarnych opartych o wartości poszczególnych zmiennych niezależnych. Formalizm ten został spopularyzowany przez Breimana i innych [8].

Akronim CHAID pochodzi od angielskiego określenia "Chi-squared Automatic Interaction Detector". Drzewa CHAID są jednymi z najstarszych drzew decyzyjnych, zaproponowane zostały już w

roku 1980 przez Kassa w [58]. Są to drzewa, w których występują decyzje niebinarne. Oznacza to, że z jednego węzła mogą odchodzić więcej niż dwie gałęzie. Podczas konstrukcji drzewa CHAID stosowane są uzależnione od typu zmiennej metody rozwiązywania problemu klasyfikacji. Dla zmiennych nominalnych stosowany jest test *Chi-kwadrat* do zidentyfikowania optymalnego w danym kroku podziału, natomiast w przypadku zmiennych ciągłych wykorzystywany jest test F.

6.2.3 Ocena i implementacja modelu

Do konstrukcji modelu wykorzystano 80% zebranych danych. Pozostałe 20% można było dzięki temu wykorzystać do przetestowania poprawności działania uzyskanego modelu. Oceniono zarówno model wykorzystujący drzewo C&RT jak i drzewo CHAID. Pierwszy z nich (czyli wykorzystujący C&RT) dokonywał poprawnej predykcji w $73,5\%$ pochodzących ze zbioru testowego przypadków, natomiast drugi (CHAID) w 69% przypadków. W świetle tych wyników zadecydowano o odrzuceniu modelu opartego na drzewie CHAID i stosowaniu w projekcie tylko modelu wykorzystującego drzewo C&RT.

Aby ułatwić stosowanie modelu w warunkach projektowych zaimplementowano go w postaci arkusza kalkulacyjnego z wykorzystaniem języka Visaul Basic. Arkusz ten automatycznie łączy się z bazą danych systemu śledzenia defektów oraz z serwerem odpowiedzialnym za kontrolę wersji. Odczytuje z tych źródeł informacje o liście defektów (zgłoszeń z systemu śledzenia defektów), dla których należy wykonać predykcję oraz dane niezbędne do wyliczenia wartości metryk. Następnie przeprowadza predykcję i prezentuje wyniki w postaci tabelarycznej. Aby zwiększyć użyteczność modelu zmieniono jego wyjście z binarnego na czterowartościowe. Możliwe jest zatem uzyskiwanie następujących predykcji dotyczących napraw defektów: zdecydowanie poprawna, raczej poprawna, raczej błędna i zdecydowanie błędna.

Rozdział 7

Podsumowanie

W ramach prac będących tematem niniejszej rozprawy wykonano sporo zadań o naturze typowo inżynierskiej. Nie były one głównym tematem, ale mimo to były krytyczne dla stanowiących meritum rozprawy eksperymentów. W związku z tym w podsumowaniu wydzielono dwie części. Pierwsza koncentruje się na odkryciach będących efektem przeprowadzonych badań empirycznych. Druga część natomiast opisuje narzędzia, nad którymi pracowano aby pozyskać dane a tym samym stworzyć środowisko niezbędne do przeprowadzenia eksperymentów.

7.1 Część badawcza

Do opisanych w niniejszej rozprawie eksperymentów zebrano łącznie metryki z 92 wersji 38 przemysłowych, otwartych oraz studenckich projektów. Warto tu raz jeszcze podkreślić ten fakt z uwagi na to, że prawdopodobnie nie istnieją inne badania dotyczące predykcji defektów, które wykorzystywałyby aż tak dużo danych. Porównanie liczebności wykorzystanego zbioru danych z tym co znajduje się w ogólnodostępnym repozytorium metryk *Promise* przedstawiono w Rozdziale 3.4.3.

7.1.1 Czynniki zwiększające dokładność predykcji.

W zakresie identyfikowania czynników poprawiających dokładność predykcji przebadano dziewięć różnych metryk. Badanie metryki polegało na wprowadzaniu jej do opartego na metrykach produktu modelu predykcji defektów i weryfikowaniu wpływu tej operacji na dokładność otrzymywanych predykcji. W przypadku siedmiu metryk dysponowano wystarczająco dużą ilością danych do tego, aby możliwe było uzyskanie wyników istotnych statystycznie. Zidentyfikowano trzy metryki: NDC, NML oraz LOC, których wprowadzenie do modelu predykcji defektów dało istotną statystycznie poprawę dokładności predykcji oraz jedną metrykę: IN, której wprowadzenie do modelu spowodowało statystycznie istotne pogorszenie dokładności predykcji.

Wynik taki wskazuje na słuszność pierwszej części tezy rozprawy mówiącej o istnieniu czynników, które powodują poprawę dokładności predykcji modelu opartego na metrykach produktu. Dodatkowo dla metryk, których wprowadzenie do modelu dało istotną statystycznie poprawę dokładności predykcji wyliczono siłę efektu. Pokazała ona, że zdecydowanie największe korzyści daje wprowadzenie do modelu metryki NDC.

Przeprowadzono również eksperyment, w którym równocześnie wprowadzano do modelu kilka metryk procesu, w tym te metryki procesu, które dawały istotną statystycznie poprawę dokładności predykcji. Okazało się, że takie podejście daje bardzo ograniczone korzyści. Poprawa dokładności

predykcji nie była istotna statystycznie. Płynie z tego wniosek, że automatyczne ustalanie zestawu metryk (w eksperymencie, o którym tu mowa stosowano postępującą regresję liniową) niekoniecznie musi prowadzić do wybrania optymalnego zestawu. Przy selekcji metryk warto wykorzystywać dostępną wiedzę płynącą z przeprowadzonych w tym obszarze badań.

Warto tu również podkreślić fakt wystąpienia sporej rozbieżności pomiędzy korelacjami poszczególnych metryk procesu z liczbą defektów a tym, które metryki dały istotną statystycznie poprawę dokładności predykcji. W świetle uzyskanych wyników można uznać, że korelacja z liczbą defektów nie jest optymalnym kryterium oceny użyteczności metryki w modelach predykcji defektów.

7.1.2 Predykcja międzyprojektowa

W eksperymentach dotyczących predykcji międzyprojektowej najpierw pokazano występowanie istotnych statystycznie różnic pomiędzy modelami predykcji defektów konstruowanych na podstawie projektów przemysłowych, otwartych oraz studenckich. Następnie wykorzystano metody eksploracji danych do identyfikowania klastrów projektów informatycznych. Posłużono się w tym celu siecią neuronową Kohonena oraz metodą k–średnich. Tym sposobem zidentyfikowano dziesięć różnych klastrów.

W kolejnym kroku weryfikowano metodami statystycznymi, czy można obronić istnienie tych klastrów z punktu widzenia predykcji defektów. Wykazano tym sposobem istnienie trzech spośród dziesięciu wcześniej zidentyfikowanych klastrów projektów. Charakterystykę tych klastrów porównano z wynikami uzyskiwanymi przez innych badaczy i uzyskano częściową zbieżność wyników. Poprawność uzyskanego podziału na klastry zweryfikowano przy pomocy analizy dyskryminacyjnej. Analiza ta posłużyła również do zidentyfikowania tych czynników, które odgrywały najistotniejszą rolę przy przypisywaniu wydań projektów do poszczególnych klastrów.

Eksperyment dotyczący klasteryzacji projektów w kontekście międzyprojektowej predykcji defektów wykazał zatem słuszność drugiej części tezy rozprawy mówiącej o istnieniu takich klastrów projektów informatycznych, że możliwe jest skuteczne zastosowanie jednego modelu predykcji defektów do wszystkich projektów wchodzących w skład danego klastra.

7.1.3 Tło eksperymentów

Eksperymenty dotyczące identyfikacji czynników poprawiających dokładność predykcji oraz klasteryzacji projektów w kontekście międzyprojektowej predykcji defektów zostały uzupełnione wyczerpującymi statystykami opisowymi. Statystyki te zostały umieszczone w dodatkach i obejmują takie elementy jak wartości średnie, minimalne, maksymalne oraz odchylenia standardowe wartości poszczególnych metryk produktu i procesu w poszczególnych projektach. Wyliczono i przedstawiono w dodatkach również korelacje (współczynniki korelacji liniowej Pearsona oraz współczynniki korelacji rang Spearmana) pomiędzy wartościami metryk a liczbami defektów. Korelacje, podobnie jak pozostałe statystyki, zostały wyliczone dla każdego z badanych projektów z osobna.

7.2 Część inżynierska

7.2.1 Narzędzia

W ramach prac badawczych dotyczących metryk oprogramowania oraz modeli predykcji defektów pracowano również nad narzędziami wspierającymi proces akwizycji danych niezbędnych do przeprowadzenia eksperymentów.

Przygotowano nową wersję służącego do wyliczania metryk oprogramowania programu *Ckjm*. W nowej wersji został dwukrotnie powiększony zestaw wyliczanych metryk. Aktualnie narzędzie to wyli-

cza 19 różnych metryk, w tym metryki pochodzące z zestawu Chidambera i Kemerera oraz z zestawu QMOOD.

Od podstaw natomiast opracowano narzędzie o nazwie *BugInfo*. Narzędzie to umożliwia gromadzenie danych o historii występowania defektów oraz pozwala na wyliczanie metryk procesu w oparciu o system kontroli wersji i wyrażenia regularne, które powinny być opracowywane na podstawie informacji zawartych w systemie śledzenia defektów.

Dane, które zostały przy pomocy powyższych programów zgromadzone, opublikowano w specjalnie do tego celu przygotowanym i uruchomionym portalu internetowym: *Metrics Repository*. Dzięki temu mogą zostać one wykorzystane przez innych badaczy w pracach poświęconych analizie i rozwojowi modeli predykcji defektów.

Szczegółowe informacje o wspomnianych tu programach przedstawiono w Rozdziale 3.4. Tam również umieszczono odnośniki wskazujące gdzie można uzyskać dostęp do tych programów. Zarówno *Ckjm* jak i *BugInfo* zostały opublikowane w internecie. Udostępniono gotowe do uruchomienia, skompilowane wersje programów, kompletne kody źródłowe oraz dokumentację wyjaśniającą w jaki sposób programów można używać.

7.2.2 Wdrożenie w przemyśle

Wykorzystując wyniki opisanych eksperymentów opracowano model predykcji defektów, który następnie zastosowano w jednym z projektów realizowanych przez firmę Capgemini Polska. Model był stosowany do przewidywania czy błąd w oprogramowaniu został poprawnie naprawiony. Uzyskiwane predykcje były poprawne w 73,5% przypadków, co wskazuje dużą użyteczność zaproponowanego rozwiązania. Szczegółowe omówienie postać wdrożonego modelu znajduje się w Rozdziale 6.

Bibliografia

[1] Giuliano Antoniol, Kamel Ayari, Massimiliano Di Penta, Foutse Khomh, Yann-Gaël Guéhéneuc. Is it a bug or an enhancement?: a text-based approach to classify change requests. *CASCON '08: Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research*, strony 304–318, New York, NY, USA, 2008. ACM. [cytowanie na str. 66, 91]

[2] Erik Arisholm, Lionel C. Briand. Predicting fault-prone components in a java legacy system. *ISESE '06: Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, strony 8–17, New York, NY, USA, 2006. ACM. [cytowanie na str. 23]

[3] Jagdish Bansiya, Carl G. Davis. A hierarchical model for object-oriented design quality assessment. *IEEE Transaction on Software Engineering*, 28(1):4–17, 2002. [cytowanie na str. 7, 13]

[4] Victor R. Basili, Lionel C. Briand, Walcélio L. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Trans. Softw. Eng.*, 22:751–761, Październik 1996. [cytowanie na str. 8]

[5] Kent Beck. *Test Driven Development: By Example*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. [cytowanie na str. 4]

[6] Robert M. Bell, Thomas J. Ostrand, Elaine J. Weyuker. Looking for bugs in all the right places. *ISSTA '06: Proceedings of the 2006 International Symposium on Software Testing and Analysis*, strony 61–72, New York, NY, USA, 2006. ACM. [cytowanie na str. 16, 18, 20, 22]

[7] Barry W. Boehm, Philip N. Papaccio. Understanding and controlling software costs. *IEEE Transaction on Software Engineering*, 14:1462–1477, Październik 1988. [cytowanie na str. 9]

[8] Leo Breiman, Jerome Friedman, Charles J. Stone, R. A. Olshen. *Classification and Regression Trees*. Chapman & Hall/CRC, wydanie 1, Styczeń 1984. [cytowanie na str. 100]

[9] Lionel C. Briand, John W. Daly, Jürgen K. Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transaction on Software Engineering*, 25:91–121, Styczeń 1999. [cytowanie na str. 8]

[10] Lionel C. Briand, Jürgen Wüst, John W. Daly, D. Victor Porter. Exploring the relationship between design measures and software quality in object-oriented systems. *Journal of Systems and Software*, 51:245–273, Maj 2000. [cytowanie na str. 8, 26]

[11] Lionel C. Briand, Jürgen Wüst, Hakim Lounis. Replicated case studies for investigating quality factors in object-oriented designs. *Empirical Software Engineering*, 6:11–58, 2001. [cytowanie na str. 8]

[12] Cagatay Catal, Banu Diri. Review: A systematic review of software fault prediction studies. *Expert Systems with Applications*, 36(4):7346–7354, 2009. [cytowanie na str. 8]

[13] Shyam R. Chidamber, Chris F. Kemerer. A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, 20(6):476–493, 1994. [cytowanie na str. 7, 12, 14, 33]

[14] Jacob Willem Cohen. *Statistical Power Analysis for the Behavioral Sciences*. Lawrence Erlbaum Associates, New York, USA, wydanie 2, 1988. [cytowanie na str. 55]

[15] Philip B Crosby. *Quality is free: The art of making quality certain*. McGraw Hill Custom Publishing, New York, USA, 1979. [cytowanie na str. 3]

[16] Marco D'Ambros, Alberto Bacchelli, Michele Lanza. On the impact of design flaws on software defects. *QSIC '10: Proceedings of the 2010 10th International Conference on Quality Software*, strony 23–31, Washington, DC, USA, 2010. IEEE Computer Society. [cytowanie na str. 65, 66, 91]

[17] Marco D'Ambros, Michele Lanza, Romain Robbes. An extensive comparison of bug prediction approaches. *MSR '10: Proceedings of the 2010 7th IEEE Working Conference on Mining Software Repositories*, strony 31–41, Washington, DC, USA, 2010. IEEE Computer Society. [cytowanie na str. 66, 91]

[18] Giovanni Denaro, Mauro Pezzè. An empirical evaluation of fault-proneness models. *Proceedings of the 24th International Conference on Software Engineering*, ICSE '02, strony 241–251, New York, NY, USA, 2002. ACM. [cytowanie na str. 9]

[19] Kalhed El Emam, Saïda Benlarbi, Nishith Goel, Shesh N. Rai. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transaction on Software Engineering*, 27(7):630–650, 2001. [cytowanie na str. 26]

[20] William M. Evanco. Comments on the confounding effect of class size on the validity of object-oriented metrics. *IEEE Transaction on Software Engineering*, 29:670–672, July 2003. [cytowanie na str. 26]

[21] Norman Fenton. When a software measure is not a measure. *Software Engineering Journal*, 7(5):357–362, 1992. [cytowanie na str. 6]

[22] Norman Fenton. Software measurement: A necessary scientific basis. *IEEE Transaction on Software Engineering*, 20:199–206, March 1994. [cytowanie na str. 8]

[23] Norman Fenton, Martin Neil, William Marsh, Peter Hearty, Lukasz Radlinski, Paul Krause. Project data incorporating qualitative factors for improved software defect prediction. *ICSEW '07: Proceedings of the 29th International Conference on Software Engineering Workshops*, strona 69, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 16]

[24] Norman E. Fenton, Martin Neil. A critique of software defect prediction models. *IEEE Transaction on Software Engineering*, 25(5):675–689, 1999. [cytowanie na str. 8, 15, 25, 26]

[25] Norman E. Fenton, Niclas Ohlsson. Quantitative analysis of faults and failures in a complex software system. *IEEE Transaction on Software Engineering*, 26:797–814, Sierpień 2000. [cytowanie na str. 16]

[26] Michael Fischer, Martin Pinzger, Harald Gall. Populating a release history database from version control and bug tracking systems. *ICSM '03: Proceedings of the International Conference on Software Maintenance*, strona 23, Washington, DC, USA, 2003. IEEE Computer Society. [cytowanie na str. 66, 91]

[27] Mariusz Flasiński. *Zarządzanie projektami informatycznymi*. Wydawnictwo Naukowe PWN SA, Warszawa, Polska, 2006. [cytowanie na str. 4]

[28] Gregory Gay, Tim Menzies, Bojan Cukic, Burak Turhan. How to build repeatable experiments. *Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, PROMISE '09, strony 15:1–15:9, New York, NY, USA, 2009. ACM. [cytowanie na str. 8]

[29] Todd L. Graves, Alan F. Karr, J. S. Marron, Harvey Siy. Predicting fault incidence using software change history. *IEEE Transaction on Software Engineering*, 26(7):653–661, 2000. [cytowanie na str. 17, 18, 19, 20, 21, 22, 23]

[30] Frank M. Gryna. *Quality Planning and Analysis: From Product Development Through Use*. McGraw Hill Custom Publishing, New York, USA, 2001. [cytowanie na str. 3]

[31] Lan Guo, Yan Ma, Bojan Cukic, Harshinder Singh. Robust prediction of fault-proneness by random forests. *Proceedings of the 15th International Symposium on Software Reliability Engineering*, strony 417–428, Washington, DC, USA, 2004. IEEE Computer Society. [cytowanie na str. 8]

[32] Janusz Górski. *Extending safety analysis techniques with formal semantics*, wolumen Technology and Assessment of Safety Critical Systems, strony 147–163. Springer Verlag, London, UK, 1994. [cytowanie na str. 5]

[33] Janusz Górski. *Inżynieria Oprogramowania w projekcie informatycznym*. Zakład Nauczania Informatyki MIKOM, Warszawa, Polska, 1999. [cytowanie na str. 4, 5, 6]

[34] Janusz Górski, Jan Magott, Adam Wardziński. Modelling fault tree using petri nets. *SAFECOMP'95: Proceedings of the 14th International Conference on Computer Safety, Reliability and Security*, strony 90–100, Belgirate, ITaly, 1995. [cytowanie na str. 5]

[35] Tracy Hall, David Bowes, Gernot Liebchen, Paul Wernick. *Evaluating Three Approaches to Extracting Fault Data from Software Change Repositories*, wolumen Product-Focused Software Process Improvement serii *Lecture Notes in Computer Science*, strony 107–115. Springer Berlin / Heidelberg, 2010. [cytowanie na str. 66, 91]

[36] Maurice H. Halstead. *Elements of Software Science (Operating and programming systems series)*. Elsevier Science Inc., New York, NY, USA, 1977. [cytowanie na str. 6, 11, 26]

[37] Ahmed E. Hassan. Predicting faults using the complexity of code changes. *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, strony 78–88, Washington, DC, USA, 2009. IEEE Computer Society. [cytowanie na str. 21]

[38] Brian Henderson-Sellers. *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1996. [cytowanie na str. 6, 7, 14]

[39] IEEE-SA Standards Board. *IEEE Std 1061-1998, Standard for a Software Quality Metrics MEthodology*. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, 1998. [cytowanie na str. 6]

[40] IEEE-SA Standards Board. *IEEE Std 829-2008, Standard for Software and System Test Documentation*. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, 2008. [cytowanie na str. 4]

[41] IEEE Standards Board. *IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software*. The Institute of Electrical and Electronics Engineers, Inc., New York, NY, USA, 1989. [cytowanie na str. 7]

[42] Timea Illes-Seifert, Barbara Paech. Exploring the relationship of a file's history and its fault-proneness: An empirical method and its application to open source programs. *Information and Software Technology*, 52(5):539–558, 2010. [cytowanie na str. 17, 19, 20, 22, 24, 25, 50, 66]

[43] Ho-Won Jung, Seung-Gweon Kim, Chang-Shin Chung. Measuring software product quality: A survey of iso/iec 9126. *IEEE Software*, 21:88–92, 2004. [cytowanie na str. 3]

[44] Marian Jureczko. *The level of agility in the testing process in a large scale financial software project*, wolumen Inżynieria oprogramowania - metody wytwarzania i wybrane zastosowania, strony 139–152. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Polska, 2008. [cytowanie na str. 4]

[45] Marian Jureczko. *Model Rayleigha w zwinnych metodykach wytwarzania oprogramowania*, wolumen Inżynieria oprogramowania - od teorii do praktyki, strony 83–92. Wydawnictwa Komunikacji i Łączności, Warszawa, Polska, 2008. [cytowanie na str. 5]

[46] Marian Jureczko, Lech Madeyski. *Predykcja defektów na podstawie metryk oprogramowania – identyfikacja klas projektów*, wolumen Inżynieria Oprogramowania w Procesach Integracji Systemów Informatycznych, strony 185–192. Wydawnictwo Komunikacji i Łączności, Gdańsk, Poland, 2010. [cytowanie na str. 5, 50, 65, 66, 71]

[47] Marian Jureczko, Lech Madeyski. Towards identifying software project clusters with regard to defect prediction. *PROMISE'2010: Proceedings of the 6th International Conference on Predictor Models in Software Engineering*. ACM, 2010. [cytowanie na str. 50, 51, 66, 71]

[48] Marian Jureczko, Lech Madeyski. Cross-project defect prediction: an empirical study. *Preprint submited to Cybernetics and Systems*, 2011. [cytowanie na str. 5, 50, 66, 71]

[49] Marian Jureczko, Lech Madeyski. Empirical validation of factors for improving software defect prediction. *Preprint submited to Information and Software Technology*, 2011. [cytowanie na str. 50, 53]

[50] Marian Jureczko, Jan Magott. *High-level Petri nets model for XP methodology*, wolumen Software engineering in progress, strony 93–108. Nakom, Poznań, Polska, 2007. [cytowanie na str. 4]

[51] Marian Jureczko, Michał Młynarski. *Zautomatyzowane testy akceptacyjne dla aplikacji internetowych w programowaniu sterowanym testami*, wolumen Od modelu do wdrożenia: kierunki badań i zastosowań inżynierii oprogramowania, strony 294–305. Wydawnictwa Komunikacji i Łączności, Warszawa, Polska, 2009. [cytowanie na str. 4]

[52] Marian Jureczko, Michał Młynarski. Automated acceptance testing tools for web applications using test-driven development. *Electrotechnical Review*, 86(09):198–202, 2010. [cytowanie na str. 4]

[53] Marian Jureczko, Paweł Skrobanek. *Oprogramowanie wspomagające analizę drzew niezdatności z zależnościami czasowymi*, wolumen Systemy czasu rzeczywistego: postępy badań i zastosowania, strony 255–267. Wydawnictwa Komunikacji i Łączności, Warszawa, Polska, 2009. [cytowanie na str. 6]

[54] Marian Jureczko, Paweł Skrobanek. Tool for analysis of the fault tree with time dependencies. *Electrotechnical Review*, 86(09):179–183, 2010. [cytowanie na str. 6]

[55] Marian Jureczko, Diomidis Spinellis. *Using Object-Oriented Design Metrics to Predict Software Defects*, wolumen Models and Methodology of System Dependability serii *Monographs of System Dependability*, strony 69–81. Oficyna Wydawnicza Politechniki Wroclawskiej, Wroclaw, Poland, 2010. [cytowanie na str. 50, 53]

[56] Vigdis By Kampenes, Tore Dybå, Jo E. Hannay, Dag I. K. Sjøberg. Systematic review: A systematic review of effect size in software engineering experiments. *Information and Software Technology*, 49(11-12):1073–1086, 2007. [cytowanie na str. 55, 56, 59, 63, 68]

[57] Stephen H. Kan. *Metrics and Models in Software Quality Engineering*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002. [cytowanie na str. 3, 4, 5, 12]

[58] G.V. Kass. Exploratory technique for investigating large quantities of categorical data. *Applied Statistics*, 29:119–127, 1980. [cytowanie na str. 101]

[59] Taghi M. Khoshgoftaar, Edward B. Allen, Robert Halstead, Gary P. Trio, Ronald M. Flass. Using process history to predict software quality. *Computer*, 31(4):66–72, 1998. [cytowanie na str. 21, 22, 23]

[60] Taghi M. Khoshgoftaar, Edward B. Allen, Kalai S. Kalaichelvan, Nishith Goel. Early quality prediction: A case study in telecommunications. *IEEE Softw.*, 13:65–71, January 1996. [cytowanie na str. 16, 18, 22]

[61] Taghi M. Khoshgoftaar, Abhijit S. Pandya, Hemant B. More. A neural network approach for predicting software development faults. *Proceedings of the Third International Symposium on Software Reliability Engineering*, strony 83–89, Research Triangle Park, NC , USA, 1992. IEEE Computer Society. [cytowanie na str. 8]

[62] Taghi M. Khoshgoftaar, Robert M. Szabo. Using neural networks to predict software faults during testing. *IEEE Transactions on Reliability*, 45:456–462, September 1996. [cytowanie na str. 8]

[63] Sunghun Kim, Thomas Zimmermann, E. James Whitehead Jr., Andreas Zeller. Predicting faults from cached history. *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, strony 489–498, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 23]

[64] Barbara Kitchenham. What's up with software metrics? - a preliminary mapping study. *Journal of Systems and Software*, 83(1):37–51, 2010. [cytowanie na str. 8, 25]

[65] Patrick Knab, Martin Pinzger, Abraham Bernstein. Predicting defect densities in source code files with decision tree learners. *MSR '06: Proceedings of the 2006 International Workshop on Mining Software Repositories*, strony 119–125, New York, NY, USA, 2006. ACM. [cytowanie na str. 18]

[66] A. Gunes Koru, Hongfang Liu. Building effective defect prediction models in practice. *IEEE Software*, 22(6):23–29, 2005. [cytowanie na str. 8, 25, 26]

[67] A. Gunes Koru, Dongsong Zhang, Hongfang Liu. Modeling the effect of size on defect proneness for open-source software. *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, strony 1–10, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 26]

[68] Arkadiusz Kumpin, Marian Jureczko. *AuTester a framework for automated teesting and test management*, wolumen Models and Methodology of System Dependability serii *Monographs of System Dependability*, strony 97–108. Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, Poland, 2010. [cytowanie na str. 4]

[69] Lucas Layman, Gunnar Kudrjavets, Nachiappan Nagappan. Iterative identification of fault-prone binaries using in-process metrics. *ESEM '08: Proceedings of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, strony 206–212, New York, NY, USA, 2008. ACM. [cytowanie na str. 20, 21]

[70] Daniel Lazar, Marian Jureczko. Bug scoring Werkzeug. Raport instytutowy, Capgemini Polska, Wrzesień 2010. [cytowanie na str. 97, 98, 100]

[71] Stefan Lessmann, Bart Baesens, Christophe Mues, Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *IEEE Transaction on Software Engineering*, 34:485–496, Lipiec 2008. [cytowanie na str. 8]

[72] Lech Madeyski. *Test-Driven Development: An Empirical Evaluation of Agile Practice*. Springer, (Heidelberg, Dordrecht, London, New York), 2010. [cytowanie na str. 26, 55]

[73] Robert Martin. Oo design quality metrics - an analysis of dependencies. *OOPSLA'94: Proceedings of Workshop Pragmatic andTheoretical Directions in Object-Oriented Software Metrics*, strony 1–8, 1994. [cytowanie na str. 14]

[74] Shinsuke Matsumoto, Yasutaka Kamei, Akito Monden, Ken ichi Matsumoto, Masahide Nakamura. An analysis of developer metrics for fault prediction. *PROMISE '10: Proceedings of the Sixth International Conference on Predictor Models in Software Engineering*. ACM, 2010. [cytowanie na str. 19, 20]

[75] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2:308–320, 1976. [cytowanie na str. 6, 15]

[76] Thilo Mende, Rainer Koschke. Revisiting the evaluation of defect prediction models. *PROMISE '09: Proceedings of the 5th International Conference on Predictor Models in Software Engineering*, strony 1–10, New York, NY, USA, 2009. ACM. [cytowanie na str. 25]

[77] Tim Menzies, Jeremy Greenwald, Art Frank. Data mining static code attributes to learn defect predictors. *IEEE Transaction on Software Engineering*, 33:2–13, January 2007. [cytowanie na str. 8, 16]

[78] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, Ayse Bener. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17:375–407, 2010. [cytowanie na str. 16]

[79] Raimund Moser, Witold Pedrycz, Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, strony 181–190, New York, NY, USA, 2008. ACM. [cytowanie na str. 17, 18, 19, 20, 21, 22, 24]

[80] Glenford J. Myers, Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. [cytowanie na str. 4]

[81] Nachiappan Nagappan, Thomas Ball. Use of relative code churn measures to predict system defect density. *ICSE '05: Proceedings of the 27th International Conference on Software engineering*, strony 284–292, New York, NY, USA, 2005. ACM. [cytowanie na str. 21]

[82] Nachiappan Nagappan, Thomas Ball. Using software dependencies and churn metrics to predict field failures: An empirical case study. *ESEM '07: Proceedings of the First International Symposium on Empirical Software Engineering and Measurement*, strony 364–373, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 18, 21]

[83] Nachiappan Nagappan, Thomas Ball, Andreas Zeller. Mining metrics to predict component failures. *ICSE '06: Proceedings of the 28th International Conference on Software Engineering*, strony 452–461, New York, NY, USA, 2006. ACM. [cytowanie na str. 28]

[84] Nachiappan Nagappan, Brendan Murphy, Victor Basili. The influence of organizational structure on software quality: an empirical case study. *ICSE '08: Proceedings of the 30th International Conference on Software Engineering*, strony 521–530, New York, NY, USA, 2008. ACM. [cytowanie na str. 20, 21]

[85] Nachiappan Nagappan, Andreas Zeller, Thomas Zimmermann, Kim Herzig, Brendan Murphy. Change bursts as defect predictors. Raport instytutowy, Microsoft Research, Luty 2010. [cytowanie na str. 18, 21]

[86] Allen P. Nikora, John C. Munson. The effects of fault counting methods on fault model quality. *Proceedings of the 28th Annual International Computer Software and Applications Conference - Volume 01*, COMPSAC '04, strony 192–201, Washington, DC, USA, 2004. IEEE Computer Society. [cytowanie na str. 8]

[87] Thomas J. Ostrand, Elaine J. Weyuker. The distribution of faults in a large industrial software system. *ISSTA '02: Proceedings of the 2002 ACM SIGSOFT International Symposium on Software Testing and Analysis*, strony 55–64, New York, NY, USA, 2002. ACM. [cytowanie na str. 16, 18, 20, 21, 22, 23]

[88] Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell. Where the bugs are. *SIGSOFT Software Engineering Notes*, 29(4):86–96, 2004. [cytowanie na str. 16, 17, 18, 20, 21, 22, 65]

[89] Thomas J. Ostrand, Elaine J. Weyuker, Robert M. Bell. Predicting the location and number of faults in large software systems. *IEEE Trans. Softw. Eng.*, 31(4):340–355, 2005. [cytowanie na str. 8, 23, 27]

[90] Sandeep Purao, Vijay Vaishnavi. Product metrics for object-oriented systems. *ACM Computing Surveys*, 35(2):191–221, 2003. [cytowanie na str. 8]

[91] Ranjith Purushothaman, Dewayne E. Perry. Toward understanding the rhetoric of small source code changes. *IEEE Transactions on Software Engineering*, 31:511–526, 2005. [cytowanie na str. 20]

[92] Jacek Ratzinger, Martin Pinzger, Harald Gall. *EQ-Mine: Predicting Short-Term Defects for Software Evolution*, wolumen Fundamental Approaches to Software Engineering serii *Lecture Notes in Computer Science*, strony 12–26. Springer Berlin / Heidelberg, 2007. [cytowanie na str. 20, 21, 24]

[93] Adrian Schröter, Thomas Zimmermann, Rahul Premraj, Andreas Zeller. If your bug database could talk. *ESEM'05: Proceedings of the 5th International Symposium on Empirical Software Engineering, Volume II: Short Papers and Posters*, strony 18–20, 2006. [cytowanie na str. 17, 19, 20, 24]

[94] Panagiotis Sfetsos, Pagagiotis Sfetsos. *Agile Software Development Quality Assurance*. IGI Publishing, Hershey, PA, USA, 2007. [cytowanie na str. 4]

[95] Emad Shihab, Zhen Ming Jiang, Walid M. Ibrahim, Bram Adams, Ahmed E. Hassan. Understanding the impact of code and process metrics on post-release defects: a case study on the eclipse project. *ESEM '10: Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, strony 1–10, New York, NY, USA, 2010. ACM. [cytowanie na str. 17, 24]

[96] Paweł Skrobanek. *Analiza zależności czasowych w drzewach niezdatności systemów związanych z bezpieczeństwem*. Praca doktorska, Politechnika Wrocławska, Wrocław, Polska, 2005. [cytowanie na str. 5, 6]

[97] Jacek Śliwerski, Thomas Zimmermann, Andreas Zeller. When do changes induce fixes? *MSR '05: Proceedings of the 2005 International Workshop on Mining Software Repositories*, strony 1–5, New York, NY, USA, 2005. ACM. [cytowanie na str. 21, 24, 25]

[98] Andreas Spillner, Tilo Linz, Hans Schaefer. *Software Testing Foundations: A Study Guide for the Certified Tester Exam*. Rocky Nook, 2007. [cytowanie na str. 4]

[99] Andrzej Stanisz. *Przystępny kurs statystyki z zastosowaniem Statistica PL na przykładach z medycyny*, wolumen 1. Statystyki podstawowe. StatSoft Polska Sp. z o. o., Kraków, Polska, 2007. [cytowanie na str. 66, 91]

[100] Andrzej Stanisz. *Przystępny kurs statystyki z zastosowaniem Statistica PL na przykładach z medycyny*, wolumen 2. Modele liniowe i nieliniowe. StatSoft Polska Sp. z o. o., Kraków, Polska, 2007. [cytowanie na str. 66, 91]

[101] Andrzej Stanisz. *Przystępny kurs statystyki z zastosowaniem Statistica PL na przykładach z medycyny*, wolumen 3. Analizy wielowymiarowe. StatSoft Polska Sp. z o. o., Kraków, Polska, 2007. [cytowanie na str. 66, 74, 75, 91]

[102] Ramanath Subramanyam, M.S. Krishnan. Empirical analysis of ck metrics for object-oriented design complexity: Implications for software defects. *IEEE Transactions on Software Engineering*, 29:297–310, 2003. [cytowanie na str. 26]

[103] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen. An empirical study on object-oriented metrics. *METRICS '99: Proceedings of the 6th International Symposium on Software Metrics*, strony 242–249, Washington, DC, USA, 1999. IEEE Computer Society. [cytowanie na str. 13]

[104] Mie Mie Thet Thwin, Tong-Seng Quah. Application of neural networks for software quality prediction using object-oriented metrics. *Journal of Systems and Software*, 76(2):147 – 156, 2005. [cytowanie na str. 8]

[105] Jeff Tian, Anthony Nguyen, Curt Allen, Ravi Appan. Experience with identifying and characterizing problem-prone modules in telecommunication software systems. *Journal of Systems and Software*, 57(3):207 – 215, 2001. [cytowanie na str. 25]

[106] William Trochim, James P. Donnelly. *Research Methods Knowledge Base*. Atomic Dog Publishing, wydanie 3, 2006. [cytowanie na str. 65]

[107] Burak Turhan, Ayse Bener, Tim Menzies. *Regularities in Learning Defect Predictors*, wolumen Product-Focused Software Process Improvement serii *Lecture Notes in Computer Science*, strony 116–130. Springer Berlin / Heidelberg, 2010. [cytowanie na str. 28]

[108] Burak Turhan, Tim Menzies, Ayşe B. Bener, Justin Di Stefano. On the relative value of cross-company and within-company data for defect prediction. *Empirical Softw. Engg.*, 14(5):540–578, 2009. [cytowanie na str. 28]

[109] Dindin Wahyudin, Rudolf Ramler, Stefan Biffl. A framework for defect prediction in specific software project contexts. *CEE-SET'08: Proceedings of the 3rd IFIP TC2 Central and East European Conference on Software Engineering Techniques*, strony 295–308. Springer LNCS, 2008. [cytowanie na str. 27]

[110] Dindin Wahyudin, Alexander Schatten, Dietmar Winkler, A. Min Tjoa, Stefan Biffl. Defect prediction using combined product and project metrics - a case study from the open source "apache" myfaces project family. *SEAA '08: Proceedings of the 2008 34th Euromicro Conference Software Engineering and Advanced Applications*, strony 207–215, Washington, DC, USA, 2008. IEEE Computer Society. [cytowanie na str. 23]

[111] Shinya Watanabe, Haruhiko Kaiya, Kenji Kaijiri. Adapting a fault prediction model to allow inter languagereuse. *PROMISE '08: Proceedings of the 4th International Workshop on Predictor Models in Software Engineering*, strony 19–24, New York, NY, USA, 2008. ACM. [cytowanie na str. 27]

[112] Elaine J. Weyuker, Thomas J. Ostrand. Comparing methods to identify defect reports in a change management database. *DEFECTS '08: Proceedings of the 2008 Workshop on Defects in Large Software Systems*, strony 27–31, New York, NY, USA, 2008. ACM. [cytowanie na str. 66, 91]

[113] Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell. Adapting a fault prediction model to allow widespread usage. *PROMISE'06: Proceedings of the 4th International Workshop on Predictor models in Software Engineering*, strony 1–5, New York, NY, USA, 2006. ACM. [cytowanie na str. 9, 16, 18, 20, 22, 23, 27]

[114] Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell. Using developer information as a factor for fault prediction. *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 9, 16, 18, 20]

[115] Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell. Do too many cooks spoil the broth? using the number of developers to enhance defect prediction models. *Empirical Software Engineering*, 13(5):539–559, 2008. [cytowanie na str. 8, 9, 16, 18, 20, 22, 23, 27]

[116] Elaine J. Weyuker, Thomas J. Ostrand, Robert M. Bell. Programmer–based fault prediction. *PROMISE '10: Proceedings of the Sixth International Conference on Predictor Models in Software Engineering*. ACM, 2010. [cytowanie na str. 16, 18, 19, 20]

[117] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Bjöorn Regnell, Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000. [cytowanie na str. 64]

[118] Tze-Jie Yu, Vincent Y. Shen, Hubert E. Dunsmore. An analysis of several software defect models. *IEEE Transaction on Software Engineering*, 14(9):1261–1270, 1988. [cytowanie na str. 23]

[119] Thomas Zimmermann, Nachiappan Nagappan, Harald Gall, Emanuel Giger, Brendan Murphy. Cross-project defect prediction: a large scale experiment on data vs. domain vs. process. *ESEC/FSE '09: Proceedings of the the 7th joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, strony 91–100, New York, NY, USA, 2009. ACM. [cytowanie na str. 19, 20, 21, 28, 66, 90, 94, 95]

[120] Thomas Zimmermann, Rahul Premraj, Andreas Zeller. Predicting defects for eclipse. *PROMISE '07: Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, strona 9, Washington, DC, USA, 2007. IEEE Computer Society. [cytowanie na str. 66, 91]

Dodatki

Dodatek A

Charakterystyki wartości metryk produktu w projektach

Tabela A.1: Charakterystyka metryki WMC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	12,96	0	252	18,62	0,30	0,25
Apache Xalan	11,15	0	138	16,42	0,30	0,18
PBeans	9,05	0	91	14,31	0,57	0,33
Apache Xerces	10,84	0	131	13,14	0,29	0,09
Apache Ant	10,82	0	120	11,70	0,49	0,37
Apache Ivy	10,97	1	205	15,21	0,31	0,28
Apache Camel	8,44	0	166	10,46	0,35	0,20
Apache Forrest	5,54	1	17	3,86	0,23	0,21
Apache Log4j	7,73	0	105	8,00	0,41	0,34
Apache Lucene	9,79	1	166	11,19	0,66	0,40
Apache Poi	13,88	0	134	14,11	0,40	0,35
Apache Synapse	7,63	0	67	8,62	0,23	0,24
Apache Velocity	8,93	0	153	14,31	0,33	0,25
Ckjm	7,20	1	23	6,78	0,81	0,75
JEdit	12,78	0	413	28,40	0,38	0,32
Przemysłowy-1	6,54	0	163	8,50	0,18	0,06
Przemysłowy-2	5,23	0	347	11,01	0,08	0,00
Przemysłowy-3	4,48	0	140	5,65	0,10	0,04
Przemysłowy-4	4,29	0	136	6,08	0,20	0,08
Przemysłowy-5	3,96	0	212	5,49	0,12	-0,02
Przemysłowy-6	8,73	1	40	7,54	0,17	0,18
Studencki-1	4,52	0	11	3,17	0,34	0,32
Studencki-2	9,02	1	53	9,15	0,57	0,30
Studencki-3	13,93	1	47	10,89	0,54	0,57
Studencki-4	7,70	1	43	8,61	0,30	0,60

Ciąg dalszy tabeli na następnej stronie.

Tabela A.1: Charakterystyka metryki WMC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	18,65	1	84	21,07	0,63	0,65
Studencki-6	6,59	0	29	6,23	0,49	0,37
Studencki-7	12,77	1	86	14,91	0,27	0,22
Studencki-8	7,72	2	32	6,64	0,69	0,54
Studencki-9	10,09	0	36	7,08	-0,03	-0,12
Studencki-10	19,22	2	101	21,36	0,68	0,39
Studencki-11	9,96	1	69	17,98	-0,18	-0,24
Studencki-12	8,20	1	30	7,44	0,21	0,30
Studencki-13	16,50	1	59	15,82	0,62	0,48
Studencki-14	6,76	2	24	4,95	-0,44	-0,41
Studencki-15	4,83	1	32	5,15	0,51	0,54
Studencki-16	9,23	0	55	9,16	0,20	0,23
Studencki-17	6,64	1	32	5,47	0,85	0,60

Tabela A.2: Charakterystyka metryki DIT

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	1,69	1	6	1,05	-0,02	-0,01
Apache Xalan	2,54	1	8	1,47	0,02	0,04
PBeans	1,38	1	4	0,74	-0,13	-0,07
Apache Xerces	1,97	1	6	1,27	0,01	0,09
Apache Ant	2,45	1	7	1,38	0,04	0,05
Apache Ivy	1,81	1	6	1,28	-0,02	-0,02
Apache Camel	1,95	1	6	1,26	-0,01	0,02
Apache Forrest	2,90	1	6	1,56	-0,09	-0,15
Apache Log4j	1,67	1	7	1,13	0,08	0,11
Apache Lucene	1,78	1	5	0,88	-0,02	0,02
Apache Poi	1,77	1	6	0,74	-0,05	0,13
Apache Synapse	1,61	1	5	0,79	-0,07	-0,06
Apache Velocity	1,72	1	5	0,91	-0,12	-0,13
Ckjm	1,50	1	4	0,97	0,02	0,12
JEdit	2,61	1	8	2,05	0,07	0,08
Przemysłowy-1	1,98	1	7	1,24	0,00	0,02
Przemysłowy-2	3,53	1	8	1,79	-0,09	-0,09
Przemysłowy-3	3,16	1	7	1,48	-0,04	-0,04
Przemysłowy-4	3,32	1	9	1,80	-0,04	-0,04
Przemysłowy-5	3,62	1	8	1,59	-0,07	-0,06
Przemysłowy-6	1,39	1	5	0,68	-0,04	-0,03
Studencki-1	1,81	1	5	1,42	0,63	0,26
Studencki-2	2,78	1	7	2,00	0,21	0,11
Studencki-3	2,97	1	6	2,44	0,60	0,60
Studencki-4	2,48	1	6	1,94	0,34	0,39

Ciąg dalszy tabeli na następnej stronie.

Tabela A.2: Charakterystyka metryki DIT (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	2,30	1	6	1,82	0,58	0,34
Studencki-6	3,08	1	5	1,68	0,20	0,17
Studencki-7	2,56	1	5	1,19	-0,06	-0,05
Studencki-8	4,11	1	6	2,45	0,34	0,32
Studencki-9	1,35	1	3	0,51	0,21	0,25
Studencki-10	2,59	1	6	2,27	0,12	0,05
Studencki-11	2,56	1	4	1,22	0,37	0,47
Studencki-12	2,11	1	6	1,70	0,32	0,27
Studencki-13	1,35	1	4	0,75	-0,30	-0,32
Studencki-14	2,92	1	8	3,03	0,46	0,50
Studencki-15	2,24	1	6	1,86	0,37	0,28
Studencki-16	1,57	1	5	1,10	-0,06	-0,10
Studencki-17	2,69	1	8	2,10	0,15	0,21

Tabela A.3: Charakterystyka metryki NOC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,36	0	31	1,97	0,06	0,05
Apache Xalan	0,55	0	29	2,43	0,04	0,06
PBeans	0,17	0	5	0,75	0,61	0,23
Apache Xerces	0,43	0	52	2,87	0,04	0,16
Apache Ant	0,68	0	102	4,21	-0,01	0,09
Apache Ivy	0,36	0	19	1,38	0,02	-0,01
Apache Camel	0,53	0	39	2,52	0,15	0,10
Apache Forrest	0,15	0	6	0,87	-0,05	-0,06
Apache Log4j	0,27	0	12	1,10	0,11	0,13
Apache Lucene	0,65	0	17	1,88	0,02	0,05
Apache Poi	0,74	0	134	7,23	0,01	0,00
Apache Synapse	0,39	0	19	2,07	0,01	-0,03
Apache Velocity	0,43	0	43	2,96	0,06	0,10
Ckjm	0,00	0	0	0,00	0,00	0,00
JEdit	0,46	0	38	2,65	-0,03	-0,03
Przemysłowy-1	0,36	0	86	2,73	0,02	0,02
Przemysłowy-2	0,69	0	546	10,11	0,01	0,03
Przemysłowy-3	0,62	0	261	8,32	-0,01	0,02
Przemysłowy-4	0,67	0	467	10,24	-0,01	0,00
Przemysłowy-5	0,76	0	464	11,86	0,00	-0,02
Przemysłowy-6	0,23	0	20	1,66	-0,04	-0,06
Studencki-1	0,00	0	0	0,00	0,00	0,00
Studencki-2	0,00	0	0	0,00	0,00	0,00
Studencki-3	0,00	0	0	0,00	0,00	0,00
Studencki-4	0,33	0	6	1,34	-0,06	-0,02

Ciąg dalszy tabeli na następnej stronie.

Tabela A.3: Charakterystyka metryki NOC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,02	0	1	0,15	-0,08	-0,11
Studencki-6	0,66	0	9	1,99	-0,08	-0,10
Studencki-7	0,69	0	10	2,46	-0,14	-0,11
Studencki-8	0,06	0	1	0,24	-0,23	-0,29
Studencki-9	0,31	0	18	1,95	-0,07	-0,08
Studencki-10	0,15	0	4	0,77	-0,07	-0,08
Studencki-11	0,22	0	3	0,80	-0,14	-0,15
Studencki-12	0,00	0	0	0,00	0,00	0,00
Studencki-13	0,10	0	1	0,31	-0,30	-0,37
Studencki-14	0,00	0	0	0,00	0,00	0,00
Studencki-15	0,10	0	4	0,62	-0,09	-0,10
Studencki-16	0,16	0	23	1,60	-0,03	-0,06
Studencki-17	0,67	0	7	1,86	-0,05	0,12

Tabela A.4: Charakterystyka metryki CBO

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	7,57	0	109	11,10	0,30	0,29
Apache Xalan	12,77	0	173	18,30	0,16	0,06
PBeans	4,75	0	35	5,09	0,57	0,37
Apache Xerces	5,57	0	60	8,53	0,38	0,28
Apache Ant	10,98	0	499	22,29	0,24	0,32
Apache Ivy	12,12	1	154	15,05	0,21	0,25
Apache Camel	10,64	0	448	20,34	0,32	0,18
Apache Forrest	10,33	2	24	5,63	0,24	0,20
Apache Log4j	7,20	0	65	9,10	0,41	0,36
Apache Lucene	10,25	0	128	11,57	0,37	0,36
Apache Poi	9,17	0	214	17,45	0,22	0,23
Apache Synapse	12,77	0	88	11,12	0,27	0,30
Apache Velocity	10,68	0	80	12,18	0,27	0,27
Ckjm	7,40	1	21	5,91	0,11	0,32
JEdit	13,34	0	346	20,96	0,39	0,25
Przemysłowy-1	12,51	0	274	16,20	0,29	0,21
Przemysłowy-2	16,30	0	860	22,76	0,26	0,13
Przemysłowy-3	16,06	0	319	16,91	0,20	0,17
Przemysłowy-4	14,42	0	601	23,56	0,26	0,19
Przemysłowy-5	16,89	0	725	24,81	0,39	0,20
Przemysłowy-6	10,21	0	76	8,82	0,15	0,18
Studencki-1	3,96	0	10	2,74	0,38	0,32
Studencki-2	7,09	0	47	7,05	0,55	0,37
Studencki-3	7,48	0	42	8,44	0,30	0,49
Studencki-4	6,30	0	27	6,06	0,30	0,59

Ciąg dalszy tabeli na następnej stronie.

Tabela A.4: Charakterystyka metryki CBO (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	10,40	0	45	11,51	0,73	0,74
Studencki-6	4,91	0	21	4,22	0,05	0,15
Studencki-7	7,21	0	21	4,72	-0,14	-0,24
Studencki-8	6,89	0	27	6,43	0,67	0,73
Studencki-9	12,06	0	69	10,60	0,00	0,07
Studencki-10	6,96	1	25	5,61	0,71	0,54
Studencki-11	6,70	1	25	6,56	-0,13	-0,06
Studencki-12	6,47	0	25	4,83	0,08	0,26
Studencki-13	6,30	0	16	3,96	0,48	0,55
Studencki-14	7,40	3	13	2,97	0,27	0,29
Studencki-15	8,19	0	33	5,79	0,40	0,32
Studencki-16	7,56	0	74	8,86	0,20	0,28
Studencki-17	7,24	1	24	4,37	0,26	0,36

Tabela A.5: Charakterystyka metryki RFC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	33,47	0	511	44,98	0,45	0,32
Apache Xalan	29,53	0	428	37,24	0,38	0,22
PBeans	25,51	0	266	44,84	0,59	0,31
Apache Xerces	20,86	0	297	31,03	0,43	0,18
Apache Ant	33,63	0	288	33,90	0,58	0,44
Apache Ivy	32,83	1	583	45,79	0,37	0,32
Apache Camel	20,87	0	322	23,12	0,34	0,20
Apache Forrest	26,60	1	67	16,23	0,22	0,16
Apache Log4j	23,59	0	321	23,25	0,34	0,30
Apache Lucene	23,96	1	392	27,38	0,68	0,43
Apache Poi	29,86	0	390	34,41	0,53	0,38
Apache Synapse	29,28	0	172	24,97	0,44	0,36
Apache Velocity	23,07	0	250	26,94	0,41	0,29
Ckjm	26,90	1	72	19,57	0,57	0,58
JEdit	39,48	0	540	56,79	0,51	0,37
Przemysłowy-1	26,81	0	346	33,66	0,30	0,18
Przemysłowy-2	24,81	0	393	26,19	0,28	0,13
Przemysłowy-3	23,84	0	355	24,51	0,20	0,14
Przemysłowy-4	24,00	0	309	26,59	0,34	0,17
Przemysłowy-5	21,29	0	293	18,54	0,24	0,18
Przemysłowy-6	24,24	1	154	22,29	0,21	0,20
Studencki-1	17,07	0	56	12,85	0,70	0,60
Studencki-2	20,46	2	105	20,41	0,53	0,34
Studencki-3	35,00	2	95	27,32	0,73	0,74
Studencki-4	27,24	1	150	30,15	0,36	0,63

Ciąg dalszy tabeli na następnej stronie.

Tabela A.5: Charakterystyka metryki RFC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	57,93	2	313	76,87	0,75	0,80
Studencki-6	12,59	0	66	12,94	0,49	0,39
Studencki-7	23,31	2	157	28,90	0,31	0,37
Studencki-8	42,83	3	118	27,82	0,61	0,46
Studencki-9	27,44	0	140	21,49	-0,01	0,00
Studencki-10	44,59	4	231	48,55	0,66	0,40
Studencki-11	21,63	2	70	18,81	0,14	0,09
Studencki-12	28,76	1	109	26,85	0,29	0,39
Studencki-13	42,25	7	92	26,29	0,71	0,71
Studencki-14	17,24	3	49	12,89	0,23	0,15
Studencki-15	20,12	1	167	29,23	0,55	0,57
Studencki-16	21,30	0	166	26,42	0,29	0,22
Studencki-17	27,98	2	118	23,46	0,46	0,42

Tabela A.6: Charakterystyka metryki LCOM

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	176,28	0	29258	1159,19	0,18	0,22
Apache Xalan	127,52	0	8413	587,17	0,25	0,20
PBeans	116,82	0	3549	507,15	0,50	0,26
Apache Xerces	91,20	0	7155	318,30	0,23	0,02
Apache Ant	82,99	0	6692	327,82	0,42	0,36
Apache Ivy	135,38	0	20326	942,23	0,17	0,24
Apache Camel	70,40	0	13617	416,78	0,26	0,17
Apache Forrest	11,72	0	88	20,55	0,21	0,21
Apache Log4j	37,17	0	4900	241,15	0,15	0,24
Apache Lucene	51,23	0	6747	317,37	0,56	0,18
Apache Poi	101,77	0	7059	400,98	0,31	0,31
Apache Synapse	38,38	0	1931	152,86	0,15	0,18
Apache Velocity	69,36	0	8092	468,41	0,34	0,04
Ckjm	22,80	0	183	57,32	0,78	0,80
JEdit	233,00	0	41713	1804,04	0,31	0,29
Przemysłowy-1	39,51	0	13203	228,20	0,13	0,06
Przemysłowy-2	59,56	0	59977	1102,18	0,01	-0,02
Przemysłowy-3	18,08	0	9730	197,23	0,03	0,03
Przemysłowy-4	21,56	0	8788	199,18	0,13	0,06
Przemysłowy-5	13,75	0	11323	212,93	0,07	-0,02
Przemysłowy-6	41,68	0	492	85,71	0,16	0,15
Studencki-1	3,52	0	22	5,40	0,10	0,13
Studencki-2	54,98	0	1058	159,87	0,57	0,34
Studencki-3	85,10	0	707	144,48	0,48	0,65
Studencki-4	44,06	0	685	124,92	0,18	0,70

Ciąg dalszy tabeli na następnej stronie.

Tabela A.6: Charakterystyka metryki LCOM (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	260,40	0	3022	635,64	0,68	0,57
Studencki-6	24,59	0	290	53,07	0,47	0,36
Studencki-7	142,28	0	3205	527,94	0,21	0,09
Studencki-8	33,11	0	390	89,85	0,65	0,43
Studencki-9	6,55	0	528	44,17	-0,06	-0,13
Studencki-10	255,89	0	4776	914,99	0,76	0,18
Studencki-11	192,85	0	2326	568,78	-0,17	-0,24
Studencki-12	37,36	0	333	74,21	0,18	0,17
Studencki-13	145,95	0	1541	364,23	0,49	0,19
Studencki-14	24,00	0	258	55,43	-0,37	-0,31
Studencki-15	13,36	0	350	54,60	0,40	0,39
Studencki-16	56,16	0	1417	138,68	0,07	0,21
Studencki-17	18,16	0	264	41,68	0,93	0,60

Tabela A.7: Charakterystyka metryki LCOM3

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	1,09	0,00	2,00	0,66	-0,08	-0,03
Apache Xalan	1,14	0,00	2,00	0,68	-0,06	-0,03
PBeans	1,18	0,00	2,00	0,76	-0,13	-0,12
Apache Xerces	1,45	0,00	2,00	0,66	-0,15	-0,15
Apache Ant	0,97	0,00	2,00	0,61	-0,07	0,03
Apache Ivy	1,04	0,00	2,00	0,66	-0,04	0,02
Apache Camel	1,08	0,00	2,00	0,72	-0,04	-0,02
Apache Forrest	1,01	0,00	2,00	0,61	-0,04	0,11
Apache Log4j	1,01	0,00	2,00	0,61	-0,07	0,03
Apache Lucene	0,96	0,00	2,00	0,65	-0,11	-0,12
Apache Poi	0,99	0,00	2,00	0,53	-0,10	-0,08
Apache Synapse	1,08	0,00	2,00	0,64	-0,13	-0,13
Apache Velocity	1,26	0,00	2,00	0,71	-0,21	-0,19
Ckjm	0,54	0,00	2,00	0,62	0,11	0,35
JEdit	1,04	0,00	2,00	0,61	-0,07	-0,05
Przemysłowy-1	1,33	0,00	2,00	0,60	-0,05	-0,02
Przemysłowy-2	1,36	0,00	2,00	0,54	-0,01	0,03
Przemysłowy-3	1,44	0,00	2,00	0,55	-0,01	-0,02
Przemysłowy-4	1,40	0,00	2,00	0,55	-0,07	-0,07
Przemysłowy-5	1,27	0,00	2,00	0,49	-0,02	-0,04
Przemysłowy-6	1,26	0,00	2,00	0,61	-0,04	-0,01
Studencki-1	1,31	0,45	2,00	0,68	-0,47	-0,57
Studencki-2	0,97	0,17	2,00	0,53	-0,01	0,15
Studencki-3	0,94	0,00	2,00	0,56	-0,04	0,22
Studencki-4	1,28	0,00	2,00	0,72	-0,06	-0,01

Ciąg dalszy tabeli na następnej stronie.

Tabela A.7: Charakterystyka metryki LCOM3 (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,91	0,00	2,00	0,55	-0,14	-0,02
Studencki-6	1,18	0,00	2,00	0,72	-0,08	-0,05
Studencki-7	1,21	0,00	2,00	0,64	-0,16	-0,05
Studencki-8	0,67	0,00	1,00	0,25	0,17	0,05
Studencki-9	0,84	0,33	2,00	0,46	0,05	0,23
Studencki-10	1,02	0,56	2,00	0,43	-0,18	-0,19
Studencki-11	0,99	0,00	2,00	0,79	-0,05	-0,08
Studencki-12	1,24	0,00	2,00	0,66	-0,39	-0,39
Studencki-13	0,70	0,00	2,00	0,53	-0,20	-0,21
Studencki-14	1,10	0,20	2,00	0,60	0,09	-0,10
Studencki-15	1,04	0,00	2,00	0,82	-0,22	-0,24
Studencki-16	1,33	0,00	2,00	0,61	-0,03	0,00
Studencki-17	1,13	0,00	2,00	0,68	-0,10	0,03

Tabela A.8: Charakterystyka metryki NPM

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	10,78	0	231	16,71	0,20	0,14
Apache Xalan	9,08	0	130	14,53	0,26	0,16
PBeans	6,06	0	45	6,73	0,46	0,31
Apache Xerces	8,43	0	83	9,30	0,18	0,03
Apache Ant	8,30	0	103	9,47	0,42	0,32
Apache Ivy	8,94	0	179	12,68	0,29	0,23
Apache Camel	6,85	0	157	9,45	0,33	0,21
Apache Forrest	4,31	1	14	2,85	0,10	0,21
Apache Log4j	5,23	0	48	5,59	0,48	0,36
Apache Lucene	6,69	0	88	7,49	0,58	0,33
Apache Poi	11,81	0	101	12,09	0,33	0,32
Apache Synapse	5,96	0	61	8,20	0,17	0,14
Apache Velocity	7,07	0	50	8,72	0,23	0,25
Ckjm	6,00	1	22	6,22	0,82	0,75
JEdit	7,76	0	218	15,90	0,52	0,24
Przemysłowy-1	4,70	0	162	7,14	0,10	0,01
Przemysłowy-2	3,34	0	347	10,65	0,06	0,08
Przemysłowy-3	2,64	0	140	4,42	0,06	0,07
Przemysłowy-4	2,62	0	136	5,33	0,14	0,04
Przemysłowy-5	2,66	0	212	5,05	0,13	0,04
Przemysłowy-6	7,25	0	37	7,11	0,14	0,12
Studencki-1	3,41	0	9	2,72	0,22	0,26
Studencki-2	5,00	1	26	5,36	0,16	0,08
Studencki-3	6,07	1	23	6,11	-0,19	-0,26
Studencki-4	6,15	1	26	6,28	0,36	0,61

Ciąg dalszy tabeli na następnej stronie.

Tabela A.8: Charakterystyka metryki NPM (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	10,44	1	56	12,06	-0,11	-0,05
Studencki-6	6,28	0	29	6,12	0,51	0,38
Studencki-7	12,31	1	83	14,56	0,29	0,26
Studencki-8	2,72	1	7	1,74	0,02	0,08
Studencki-9	7,16	0	34	6,61	-0,01	-0,13
Studencki-10	10,30	1	52	12,86	0,08	0,27
Studencki-11	9,44	1	67	17,87	-0,21	-0,32
Studencki-12	5,67	0	24	5,60	0,18	0,22
Studencki-13	15,05	1	59	15,67	0,59	0,53
Studencki-14	6,68	2	24	4,98	-0,46	-0,44
Studencki-15	2,74	0	12	2,18	0,06	0,12
Studencki-16	8,57	0	55	8,55	0,16	0,23
Studencki-17	5,87	1	32	5,50	0,77	0,42

Tabela A.9: Charakterystyka metryki DAM

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,57	0	1	0,47	0,18	0,17
Apache Xalan	0,43	0	1	0,47	0,06	0,03
PBeans	0,53	0	1	0,50	0,22	0,19
Apache Xerces	0,27	0	1	0,41	0,19	0,17
Apache Ant	0,67	0	1	0,43	0,14	0,11
Apache Ivy	0,63	0	1	0,46	0,09	0,07
Apache Camel	0,62	0	1	0,48	0,06	0,04
Apache Forrest	0,61	0	1	0,47	0,12	0,01
Apache Log4j	0,23	0	1	0,37	0,29	0,36
Apache Lucene	0,50	0	1	0,45	0,16	0,24
Apache Poi	0,49	0	1	0,40	0,12	0,11
Apache Synapse	0,60	0	1	0,44	0,19	0,23
Apache Velocity	0,44	0	1	0,47	0,18	0,17
Ckjm	0,89	0	1	0,31	0,22	-0,04
JEdit	0,53	0	1	0,46	0,14	0,20
Przemysłowy-1	0,38	0	1	0,41	-0,02	-0,03
Przemysłowy-2	0,15	0	1	0,33	0,03	0,04
Przemysłowy-3	0,13	0	1	0,29	0,05	0,06
Przemysłowy-4	0,12	0	1	0,27	0,04	0,05
Przemysłowy-5	0,07	0	1	0,20	-0,01	0,03
Przemysłowy-6	0,59	0	1	0,48	0,09	0,09
Studencki-1	0,07	0	1	0,22	0,07	0,09
Studencki-2	0,69	0	1	0,45	0,15	0,10
Studencki-3	0,77	0	1	0,41	0,23	0,12
Studencki-4	0,38	0	1	0,43	-0,14	-0,05

Ciąg dalszy tabeli na następnej stronie.

Tabela A.9: Charakterystyka metryki DAM (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,76	0	1	0,38	0,03	-0,13
Studencki-6	0,58	0	1	0,49	0,16	0,14
Studencki-7	0,61	0	1	0,49	0,06	0,01
Studencki-8	0,87	0	1	0,26	-0,17	-0,21
Studencki-9	0,79	0	1	0,36	-0,12	-0,18
Studencki-10	0,75	0	1	0,43	0,13	-0,05
Studencki-11	0,34	0	1	0,48	0,40	0,36
Studencki-12	0,51	0	1	0,48	0,33	0,30
Studencki-13	0,71	0	1	0,45	0,37	0,41
Studencki-14	0,71	0	1	0,46	-0,18	-0,11
Studencki-15	0,40	0	1	0,47	0,15	0,12
Studencki-16	0,54	0	1	0,49	0,04	0,04
Studencki-17	0,36	0	1	0,42	0,18	0,19

Tabela A.10: Charakterystyka metryki MOA

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,94	0	24	2,11	0,31	0,31
Apache Xalan	0,81	0	18	1,78	0,21	0,10
PBeans	0,32	0	6	0,90	0,22	0,37
Apache Xerces	0,79	0	41	2,68	0,39	0,18
Apache Ant	0,69	0	11	1,38	0,31	0,28
Apache Ivy	0,66	0	12	1,27	0,23	0,10
Apache Camel	0,66	0	9	1,20	0,21	0,11
Apache Forrest	0,40	0	3	0,76	0,17	0,19
Apache Log4j	0,82	0	14	1,45	0,21	0,14
Apache Lucene	1,18	0	13	2,02	0,37	0,18
Apache Poi	0,90	0	34	2,74	0,16	0,12
Apache Synapse	0,31	0	6	0,73	0,20	0,24
Apache Velocity	0,47	0	10	1,15	0,40	0,30
Ckjm	0,40	0	2	0,84	0,13	0,09
JEdit	1,05	0	17	2,05	0,36	0,29
Przemysłowy-1	0,17	0	83	1,02	0,05	0,02
Przemysłowy-2	0,04	0	25	0,64	0,01	0,04
Przemysłowy-3	0,07	0	158	1,76	0,05	0,04
Przemysłowy-4	0,05	0	9	0,42	-0,01	-0,02
Przemysłowy-5	0,05	0	66	1,23	0,01	-0,03
Przemysłowy-6	0,95	0	11	1,86	0,12	0,20
Studencki-1	0,33	0	2	0,62	0,55	0,36
Studencki-2	1,29	0	6	1,63	0,12	0,05
Studencki-3	1,10	0	4	1,42	0,63	0,60
Studencki-4	0,27	0	2	0,57	0,41	0,52

Ciąg dalszy tabeli na następnej stronie.

Tabela A.10: Charakterystyka metryki MOA (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,95	0	7	2,06	0,66	0,70
Studencki-6	0,31	0	3	0,66	0,41	0,31
Studencki-7	1,15	0	6	1,58	0,39	0,38
Studencki-8	1,28	0	11	2,56	0,75	0,66
Studencki-9	0,52	0	8	1,32	-0,01	0,03
Studencki-10	0,85	0	6	1,85	0,18	0,29
Studencki-11	0,89	0	10	1,95	0,74	0,53
Studencki-12	1,04	0	10	1,95	0,23	0,36
Studencki-13	1,85	0	12	2,78	0,18	0,28
Studencki-14	0,48	0	2	0,65	0,32	0,33
Studencki-15	0,71	0	6	1,20	0,05	0,29
Studencki-16	0,89	0	10	1,81	0,17	0,20
Studencki-17	0,33	0	2	0,52	0,02	-0,04

Tabela A.11: Charakterystyka metryki MFA

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,29	0,00	1,00	0,39	-0,03	-0,03
Apache Xalan	0,55	0,00	1,00	0,44	0,03	0,00
PBeans	0,23	0,00	1,00	0,40	-0,06	-0,02
Apache Xerces	0,35	0,00	1,00	0,42	0,01	0,08
Apache Ant	0,50	0,00	1,00	0,41	-0,02	-0,05
Apache Ivy	0,29	0,00	1,00	0,38	-0,03	-0,06
Apache Camel	0,40	0,00	1,00	0,41	-0,03	0,00
Apache Forrest	0,55	0,00	0,93	0,35	-0,09	-0,18
Apache Log4j	0,30	0,00	1,00	0,40	0,07	0,07
Apache Lucene	0,33	0,00	1,00	0,34	-0,04	-0,02
Apache Poi	0,33	0,00	1,00	0,30	-0,05	0,09
Apache Synapse	0,28	0,00	1,00	0,35	-0,08	-0,06
Apache Velocity	0,41	0,00	1,00	0,42	-0,10	-0,16
Ckjm	0,26	0,00	0,95	0,43	0,02	0,07
JEdit	0,49	0,00	1,00	0,44	0,01	0,03
Przemysłowy-1	0,40	0,00	1,00	0,42	-0,02	-0,01
Przemysłowy-2	0,72	0,00	1,00	0,35	-0,08	-0,06
Przemysłowy-3	0,67	0,00	1,00	0,35	-0,02	-0,05
Przemysłowy-4	0,67	0,00	1,00	0,38	-0,09	-0,08
Przemysłowy-5	0,74	0,00	1,00	0,33	-0,04	-0,04
Przemysłowy-6	0,22	0,00	1,00	0,35	-0,03	-0,03
Studencki-1	0,31	0,00	1,00	0,42	0,36	0,25
Studencki-2	0,50	0,00	1,00	0,47	0,12	0,02
Studencki-3	0,40	0,00	1,00	0,49	0,54	0,41
Studencki-4	0,46	0,00	0,99	0,43	0,13	0,25

Ciąg dalszy tabeli na następnej stronie.

Tabela A.11: Charakterystyka metryki MFA (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,39	0,00	1,00	0,44	0,36	0,23
Studencki-6	0,58	0,00	1,00	0,41	0,03	-0,08
Studencki-7	0,50	0,00	0,99	0,35	-0,04	-0,04
Studencki-8	0,64	0,00	0,99	0,47	0,28	0,23
Studencki-9	0,24	0,00	0,93	0,36	0,32	0,33
Studencki-10	0,31	0,00	0,98	0,45	0,03	-0,04
Studencki-11	0,66	0,00	1,00	0,42	0,27	0,33
Studencki-12	0,35	0,00	1,00	0,46	0,22	0,18
Studencki-13	0,18	0,00	0,90	0,34	-0,30	-0,31
Studencki-14	0,29	0,00	0,97	0,44	0,49	0,48
Studencki-15	0,37	0,00	1,00	0,45	0,16	0,21
Studencki-16	0,22	0,00	1,00	0,39	-0,10	-0,12
Studencki-17	0,41	0,00	0,99	0,38	0,03	0,03

Tabela A.12: Charakterystyka metryki CAM

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,49	0,00	1,00	0,25	-0,22	-0,25
Apache Xalan	0,48	0,00	1,00	0,27	-0,17	-0,17
PBeans	0,51	0,00	1,00	0,28	-0,33	-0,30
Apache Xerces	0,51	0,00	1,00	0,25	-0,19	-0,15
Apache Ant	0,48	0,00	1,00	0,26	-0,29	-0,35
Apache Ivy	0,49	0,06	1,00	0,25	-0,20	-0,29
Apache Camel	0,49	0,00	1,00	0,26	-0,19	-0,15
Apache Forrest	0,47	0,21	1,00	0,17	-0,15	-0,16
Apache Log4j	0,44	0,00	1,00	0,23	-0,18	-0,20
Apache Lucene	0,44	0,00	1,00	0,23	-0,28	-0,34
Apache Poi	0,41	0,00	1,00	0,20	-0,23	-0,23
Apache Synapse	0,48	0,00	1,00	0,22	-0,23	-0,27
Apache Velocity	0,47	0,00	1,00	0,22	-0,26	-0,29
Ckjm	0,47	0,17	1,00	0,25	-0,43	-0,53
JEdit	0,46	0,00	1,00	0,25	-0,23	-0,28
Przemysłowy-1	0,52	0,00	1,00	0,27	-0,15	-0,17
Przemysłowy-2	0,58	0,00	1,00	0,20	-0,13	-0,09
Przemysłowy-3	0,57	0,00	1,00	0,21	-0,09	-0,06
Przemysłowy-4	0,52	0,00	1,00	0,24	-0,12	-0,11
Przemysłowy-5	0,59	0,00	1,00	0,19	-0,10	-0,04
Przemysłowy-6	0,50	0,11	1,00	0,25	-0,15	-0,18
Studencki-1	0,45	0,00	1,00	0,24	-0,01	-0,07
Studencki-2	0,50	0,17	1,00	0,24	-0,21	-0,19
Studencki-3	0,47	0,18	1,00	0,25	-0,48	-0,62
Studencki-4	0,59	0,12	1,00	0,26	-0,43	-0,54

Ciąg dalszy tabeli na następnej stronie.

Tabela A.12: Charakterystyka metryki CAM (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,44	0,15	1,00	0,23	-0,40	-0,59
Studencki-6	0,50	0,00	1,00	0,30	-0,15	-0,21
Studencki-7	0,40	0,12	1,00	0,21	-0,22	-0,29
Studencki-8	0,52	0,31	0,75	0,15	-0,34	-0,43
Studencki-9	0,43	0,00	1,00	0,18	0,17	0,20
Studencki-10	0,43	0,11	1,00	0,25	-0,27	-0,29
Studencki-11	0,59	0,11	1,00	0,29	0,16	0,21
Studencki-12	0,56	0,19	1,00	0,28	-0,28	-0,31
Studencki-13	0,45	0,11	1,00	0,19	-0,42	-0,40
Studencki-14	0,44	0,27	0,72	0,14	-0,25	-0,17
Studencki-15	0,61	0,14	1,00	0,24	-0,45	-0,55
Studencki-16	0,48	0,00	1,00	0,28	-0,19	-0,22
Studencki-17	0,61	0,25	1,00	0,25	-0,29	-0,29

Tabela A.13: Charakterystyka metryki IC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,28	0	4	0,58	0,11	0,08
Apache Xalan	0,81	0	5	1,07	0,04	0,00
PBeans	0,22	0	2	0,50	-0,12	-0,03
Apache Xerces	0,36	0	4	0,69	0,18	0,16
Apache Ant	0,71	0	5	0,97	0,13	0,11
Apache Ivy	0,39	0	5	0,75	0,07	0,10
Apache Camel	0,38	0	4	0,58	0,03	0,02
Apache Forrest	0,60	0	2	0,70	-0,08	-0,18
Apache Log4j	0,35	0	3	0,62	0,16	0,16
Apache Lucene	0,52	0	3	0,70	0,06	0,13
Apache Poi	0,56	0	3	0,54	0,06	0,24
Apache Synapse	0,19	0	3	0,51	-0,04	0,00
Apache Velocity	0,37	0	3	0,65	-0,11	-0,14
Ckjm	0,20	0	2	0,63	-0,04	0,06
JEdit	0,64	0	5	0,98	0,12	0,09
Przemysłowy-1	0,46	0	4	0,72	0,02	0,02
Przemysłowy-2	1,39	0	5	1,06	-0,06	-0,06
Przemysłowy-3	1,27	0	5	1,07	-0,01	0,00
Przemysłowy-4	1,19	0	6	1,18	-0,02	-0,02
Przemysłowy-5	1,35	0	6	1,09	-0,05	-0,03
Przemysłowy-6	0,31	0	3	0,52	0,06	0,13
Studencki-1	0,19	0	1	0,40	-0,31	-0,36
Studencki-2	0,15	0	1	0,36	-0,03	-0,04
Studencki-3	0,00	0	0	0,00	0,00	0,00
Studencki-4	0,24	0	2	0,50	-0,02	0,04

Ciąg dalszy tabeli na następnej stronie.

Tabela A.13: Charakterystyka metryki IC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,30	0	1	0,46	-0,20	-0,08
Studencki-6	0,47	0	2	0,67	0,11	0,11
Studencki-7	0,21	0	1	0,41	0,24	0,24
Studencki-8	0,06	0	1	0,24	-0,23	-0,29
Studencki-9	0,40	0	2	0,55	0,24	0,30
Studencki-10	0,37	0	3	0,69	0,11	0,18
Studencki-11	0,07	0	1	0,27	-0,14	-0,15
Studencki-12	0,11	0	1	0,32	-0,17	-0,18
Studencki-13	0,25	0	2	0,55	-0,23	-0,20
Studencki-14	0,32	0	2	0,63	0,45	0,44
Studencki-15	0,14	0	1	0,35	-0,14	-0,13
Studencki-16	0,21	0	2	0,43	-0,10	-0,09
Studencki-17	0,18	0	2	0,44	0,00	0,08

Tabela A.14: Charakterystyka metryki CBM

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,59	0	19	1,74	0,18	0,09
Apache Xalan	2,88	0	33	4,71	0,12	0,02
PBeans	0,38	0	7	1,12	-0,14	-0,04
Apache Xerces	1,46	0	25	3,27	0,17	0,15
Apache Ant	1,23	0	19	2,13	0,13	0,10
Apache Ivy	0,66	0	18	1,64	0,06	0,10
Apache Camel	0,71	0	21	1,83	0,02	0,03
Apache Forrest	0,81	0	6	1,22	-0,09	-0,19
Apache Log4j	0,66	0	13	1,50	0,16	0,16
Apache Lucene	1,14	0	15	1,85	0,19	0,15
Apache Poi	2,46	0	20	2,81	0,06	0,24
Apache Synapse	0,34	0	7	0,98	0,00	0,01
Apache Velocity	0,54	0	9	1,06	-0,11	-0,14
Ckjm	0,20	0	2	0,63	-0,04	0,06
JEdit	1,54	0	25	3,16	0,18	0,10
Przemysłowy-1	0,91	0	22	2,20	0,00	0,01
Przemysłowy-2	2,32	0	22	2,60	-0,04	-0,05
Przemysłowy-3	1,75	0	19	1,97	-0,01	0,00
Przemysłowy-4	1,86	0	19	2,61	-0,01	-0,02
Przemysłowy-5	1,76	0	19	1,68	-0,03	-0,02
Przemysłowy-6	0,46	0	4	0,87	0,04	0,12
Studencki-1	0,19	0	1	0,40	-0,31	-0,36
Studencki-2	0,15	0	1	0,36	-0,03	-0,04
Studencki-3	0,00	0	0	0,00	0,00	0,00
Studencki-4	0,36	0	3	0,78	-0,04	0,03

Ciąg dalszy tabeli na następnej stronie.

Tabela A.14: Charakterystyka metryki CBM (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	0,30	0	1	0,46	-0,20	-0,08
Studencki-6	0,47	0	2	0,67	0,11	0,11
Studencki-7	0,23	0	2	0,48	0,26	0,25
Studencki-8	0,06	0	1	0,24	-0,23	-0,29
Studencki-9	1,70	0	11	2,83	0,46	0,38
Studencki-10	0,52	0	7	1,37	0,01	0,18
Studencki-11	0,15	0	2	0,53	-0,14	-0,15
Studencki-12	0,11	0	1	0,32	-0,17	-0,18
Studencki-13	0,55	0	5	1,39	-0,18	-0,19
Studencki-14	0,80	0	5	1,53	0,45	0,44
Studencki-15	0,33	0	3	0,90	-0,10	-0,13
Studencki-16	0,23	0	3	0,49	-0,10	-0,09
Studencki-17	0,22	0	4	0,67	-0,02	0,08

Tabela A.15: Charakterystyka metryki AMC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	25,58	0,00	894,50	46,64	0,13	0,24
Apache Xalan	57,36	0,00	1875,00	144,6	0,18	0,312
PBeans	15,81	0,00	98,62	18,54	0,38	0,26
Apache Xerces	21,56	0,00	779,80	55,30	0,11	0,22
Apache Ant	26,86	0,00	2052,00	88,12	0,03	0,29
Apache Ivy	18,35	0,00	291,33	27,43	0,12	0,24
Apache Camel	10,93	0,00	158,67	11,64	0,07	0,11
Apache Forrest	33,65	0,00	91,00	23,71	0,01	0,06
Apache Log4j	20,26	0,00	134,00	18,11	0,02	0,07
Apache Lucene	22,74	0,00	237,20	24,66	0,10	0,19
Apache Poi	18,47	0,00	616,38	36,91	0,07	0,22
Apache Synapse	26,53	0,00	187,50	24,50	0,20	0,24
Apache Velocity	21,10	0,00	276,00	28,85	0,28	0,22
Ckjm	19,27	0,00	30,62	10,38	0,15	0,34
JEdit	30,65	0,00	540,00	36,38	0,06	0,19
Przemysłowy-1	26,05	0,00	432,67	32,01	0,18	0,18
Przemysłowy-2	31,49	0,00	845,85	39,10	0,15	0,18
Przemysłowy-3	31,19	0,00	798,00	40,87	0,12	0,19
Przemysłowy-4	33,38	0,00	636,20	39,69	0,19	0,16
Przemysłowy-5	32,87	0,00	3492,00	53,07	0,13	0,22
Przemysłowy-6	17,40	0,00	198,50	21,07	0,07	0,09
Studencki-1	41,72	0,00	254,50	57,60	0,38	0,71
Studencki-2	21,40	0,00	151,00	28,69	0,14	0,17
Studencki-3	26,28	0,00	91,37	23,85	0,45	0,53
Studencki-4	19,79	0,00	106,33	20,48	0,28	0,36

Ciąg dalszy tabeli na następnej stronie.

Tabela A.15: Charakterystyka metryki AMC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	27,12	0,00	72,27	19,95	0,67	0,70
Studencki-6	6,28	0,00	58,00	8,35	0,05	0,28
Studencki-7	5,96	2,50	21,40	3,65	0,30	0,40
Studencki-8	39,00	3,00	91,25	26,22	0,07	0,17
Studencki-9	29,16	0,00	100,50	17,56	0,19	0,16
Studencki-10	19,97	3,75	63,67	13,96	0,16	0,18
Studencki-11	33,71	2,06	129,00	31,64	0,65	0,54
Studencki-12	35,88	0,00	190,00	41,12	0,05	0,26
Studencki-13	32,11	5,40	81,00	23,37	-0,07	0,12
Studencki-14	13,16	0,00	51,67	15,93	0,33	0,36
Studencki-15	27,72	0,00	95,14	27,73	0,49	0,48
Studencki-16	9,75	0,00	49,46	10,01	0,20	0,14
Studencki-17	29,55	2,00	148,00	31,89	0,18	0,15

Tabela A.16: Charakterystyka metryki Ca

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	3,86	0	109	8,90	0,12	0,17
Apache Xalan	6,07	0	156	15,44	0,11	0,04
PBeans	2,75	0	12	2,93	0,35	0,24
Apache Xerces	2,94	0	57	6,83	0,15	0,23
Apache Ant	5,71	0	498	21,83	0,14	0,08
Apache Ivy	6,26	0	147	12,32	0,09	0,07
Apache Camel	5,13	0	446	19,63	0,26	0,17
Apache Forrest	1,42	0	7	1,86	-0,17	-0,15
Apache Log4j	3,94	0	58	8,26	0,33	0,28
Apache Lucene	5,75	0	104	10,51	0,17	0,15
Apache Poi	4,77	0	212	15,41	0,05	0,04
Apache Synapse	4,22	0	88	9,91	0,01	0,10
Apache Velocity	5,50	0	76	10,80	0,10	0,08
Ckjm	2,30	0	7	2,21	0,49	0,30
JEdit	8,11	0	291	18,38	0,35	0,22
Przemysłowy-1	3,46	0	263	11,83	0,10	-0,03
Przemysłowy-2	2,71	0	860	20,00	0,19	-0,02
Przemysłowy-3	2,20	0	319	13,31	0,11	0,02
Przemysłowy-4	2,44	0	549	20,60	0,17	0,09
Przemysłowy-5	2,75	0	725	22,66	0,34	-0,05
Przemysłowy-6	4,50	0	74	7,45	0,04	0,01
Studencki-1	1,93	0	6	1,69	-0,03	0,07
Studencki-2	4,40	0	36	5,83	0,51	0,28
Studencki-3	5,59	0	39	7,86	0,21	0,48
Studencki-4	2,30	0	9	2,38	0,33	0,51

Ciąg dalszy tabeli na następnej stronie.

Tabela A.16: Charakterystyka metryki Ca (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	6,56	0	34	8,49	0,73	0,69
Studencki-6	2,25	0	17	3,40	-0,09	-0,08
Studencki-7	3,38	0	21	4,37	-0,34	-0,31
Studencki-8	3,78	0	15	3,42	0,75	0,69
Studencki-9	5,59	0	67	10,25	-0,09	-0,07
Studencki-10	5,00	0	17	4,26	0,62	0,51
Studencki-11	2,33	0	7	2,22	-0,37	-0,43
Studencki-12	3,16	0	25	3,95	0,01	0,20
Studencki-13	3,90	0	13	3,39	0,41	0,27
Studencki-14	2,16	0	8	2,98	-0,45	-0,45
Studencki-15	4,67	0	22	4,00	0,34	0,15
Studencki-16	2,89	0	71	6,63	-0,08	-0,13
Studencki-17	3,47	0	24	4,70	0,21	0,45

Tabela A.17: Charakterystyka metryki Ce

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	0,00	0	0	0,00	0,00	0,00
Apache Xalan	7,39	0	68	9,31	0,18	0,07
PBeans	2,44	0	30	4,42	0,58	0,44
Apache Xerces	2,98	0	55	5,00	0,52	0,28
Apache Ant	5,63	0	37	5,52	0,45	0,37
Apache Ivy	4,89	0	98	8,34	0,33	0,24
Apache Camel	6,13	0	76	6,51	0,25	0,12
Apache Forrest	9,31	0	24	6,22	0,25	0,22
Apache Log4j	3,62	0	51	4,01	0,31	0,24
Apache Lucene	4,99	0	46	5,64	0,51	0,37
Apache Poi	4,74	0	133	8,79	0,35	0,33
Apache Synapse	8,76	0	38	6,71	0,43	0,30
Apache Velocity	6,02	0	61	7,47	0,39	0,30
Ckjm	5,70	0	21	6,80	-0,04	-0,03
JEdit	6,78	0	101	8,19	0,42	0,28
Przemysłowy-1	9,22	0	110	10,86	0,32	0,21
Przemysłowy-2	13,62	0	130	9,77	0,22	0,13
Przemysłowy-3	13,88	0	116	10,42	0,17	0,17
Przemysłowy-4	12,00	0	76	9,98	0,28	0,17
Przemysłowy-5	14,15	0	87	9,41	0,21	0,20
Przemysłowy-6	6,00	0	38	6,88	0,17	0,15
Studencki-1	2,15	0	7	2,13	0,48	0,30
Studencki-2	3,57	0	20	4,14	0,43	0,29
Studencki-3	3,03	0	9	3,12	0,68	0,69
Studencki-4	4,39	0	26	5,43	0,29	0,57

Ciąg dalszy tabeli na następnej stronie.

Tabela A.17: Charakterystyka metryki Ce (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	7,42	0	45	11,89	0,75	0,70
Studencki-6	2,70	0	14	2,48	0,20	0,30
Studencki-7	3,87	0	13	3,04	0,25	0,24
Studencki-8	5,39	0	27	6,70	0,61	0,57
Studencki-9	6,47	0	46	6,10	0,16	0,17
Studencki-10	3,81	0	23	5,09	0,69	0,40
Studencki-11	4,85	0	23	6,13	-0,05	0,04
Studencki-12	3,96	0	16	4,11	0,15	0,27
Studencki-13	4,00	0	14	3,83	0,35	0,51
Studencki-14	5,24	0	13	4,21	0,51	0,52
Studencki-15	4,67	0	33	6,14	0,45	0,46
Studencki-16	4,71	0	36	6,49	0,34	0,32
Studencki-17	4,16	0	15	2,88	0,14	0,06

Tabela A.18: Charakterystyka metryki Max(CC)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	4,27	0	95	6,95	0,30	0,26
Apache Xalan	4,36	0	143	7,15	0,18	0,20
PBeans	2,86	0	35	5,82	0,27	0,27
Apache Xerces	3,57	0	147	8,70	0,26	0,20
Apache Ant	4,68	0	53	6,19	0,31	0,30
Apache Ivy	3,14	0	29	3,80	0,22	0,18
Apache Camel	2,17	0	33	2,61	0,20	0,18
Apache Forrest	3,12	1	17	3,43	-0,10	-0,02
Apache Log4j	3,44	0	23	3,26	0,29	0,23
Apache Lucene	4,68	0	236	14,35	0,05	0,19
Apache Poi	3,55	0	126	7,40	0,28	0,28
Apache Synapse	7,26	0	50	7,91	0,31	0,29
Apache Velocity	3,78	0	209	13,01	0,22	0,10
Ckjm	3,20	1	7	2,35	0,51	0,41
JEdit	6,72	0	167	11,61	0,25	0,31
Przemysłowy-1	4,36	0	252	7,56	0,18	0,12
Przemysłowy-2	3,10	0	85	4,93	0,17	0,12
Przemysłowy-3	3,09	0	74	4,71	0,15	0,16
Przemysłowy-4	2,83	0	121	5,66	0,15	0,02
Przemysłowy-5	2,28	0	62	3,17	0,13	0,10
Przemysłowy-6	3,38	0	47	4,35	0,07	0,18
Studencki-1	3,33	0	23	4,46	0,28	0,25
Studencki-2	2,42	0	16	3,04	0,62	0,23
Studencki-3	2,52	0	15	2,96	0,20	0,35
Studencki-4	2,64	0	22	3,89	0,25	0,19

Ciąg dalszy tabeli na następnej stronie.

Tabela A.18: Charakterystyka metryki Max(CC) (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	11,37	0	72	17,64	0,32	0,56
Studencki-6	1,19	0	6	1,11	0,03	0,17
Studencki-7	1,69	0	6	1,52	0,28	0,24
Studencki-8	2,28	1	7	1,87	0,07	0,19
Studencki-9	3,70	0	20	2,95	0,16	0,05
Studencki-10	3,85	0	14	3,30	0,16	0,18
Studencki-11	2,63	0	40	7,55	0,63	-0,08
Studencki-12	3,49	0	29	5,72	0,12	0,33
Studencki-13	6,90	0	38	9,54	0,32	0,27
Studencki-14	2,88	1	17	4,59	0,16	0,18
Studencki-15	1,76	0	8	1,81	0,19	0,41
Studencki-16	2,78	0	34	3,56	0,18	0,11
Studencki-17	2,04	0	17	2,70	0,07	0,28

Tabela A.19: Charakterystyka metryki Avg(CC)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	1,25	0,00	10,00	1,00	0,23	0,22
Apache Xalan	1,32	0,00	22,00	1,23	0,13	0,13
PBeans	0,91	0,00	5,14	0,6	0,35	0,24
Apache Xerces	1,29	0,00	21,00	1,41	0,19	0,10
Apache Ant	1,39	0,00	8,45	0,9	0,17	0,25
Apache Ivy	1,20	0,00	6,50	0,77	0,12	0,18
Apache Camel	0,94	0,00	8,50	0,61	0,14	0,14
Apache Forrest	1,28	0,50	4,00	0,82	-0,10	-0,06
Apache Log4j	1,34	0,00	6,25	0,81	0,18	0,16
Apache Lucene	1,28	0,00	16,20	1,26	0,00	0,13
Apache Poi	1,16	0,00	17,13	0,99	0,17	0,20
Apache Synapse	2,01	0,00	12,50	1,55	0,16	0,22
Apache Velocity	1,20	0,00	23,00	1,51	0,13	0,13
Ckjm	1,35	0,50	2,23	0,6	0,17	0,16
JEdit	1,83	0,00	28,67	1,80	0,16	0,25
Przemysłowy-1	1,53	0,00	22,08	1,61	0,15	0,11
Przemysłowy-2	1,26	0,00	25,50	1,50	0,13	0,11
Przemysłowy-3	1,28	0,00	26,20	1,46	0,10	0,15
Przemysłowy-4	1,07	0,00	30,50	1,43	0,09	0,03
Przemysłowy-5	1,02	0,00	17,00	1,10	0,10	0,09
Przemysłowy-6	1,26	0,00	9,00	0,94	0,05	0,15
Studencki-1	1,34	0,00	4,50	1,03	0,35	0,26
Studencki-2	1,04	0,00	2,00	0,44	0,32	0,23
Studencki-3	1,22	0,00	3,93	0,7	0,31	0,41
Studencki-4	1,06	0,00	4,50	0,77	0,58	0,29

Ciąg dalszy tabeli na następnej stronie.

Tabela A.19: Charakterystyka metryki Avg(CC) (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	1,50	0,00	3,23	0,8	0,27	0,47
Studencki-6	0,73	0,00	2,00	0,45	0,13	0,25
Studencki-7	0,91	0,00	1,80	0,29	0,26	0,28
Studencki-8	1,06	0,50	2,00	0,40	0,23	0,28
Studencki-9	1,44	0,00	3,50	0,68	0,01	-0,08
Studencki-10	1,31	0,00	4,00	0,76	0,05	0,11
Studencki-11	0,79	0,00	6,43	1,2	0,55	-0,10
Studencki-12	1,12	0,00	5,27	0,9	0,08	0,28
Studencki-13	1,75	0,00	10,47	2,18	0,09	0,24
Studencki-14	1,40	0,50	6,00	1,40	0,17	0,02
Studencki-15	0,89	0,00	3,50	0,57	0,05	0,24
Studencki-16	1,03	0,00	6,00	0,72	0,14	0,12
Studencki-17	0,95	0,00	3,50	0,63	0,15	0,40

Tabela A.20: Charakterystyka metryki LOC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Apache Tomcat	350,44	0	7956	644,84	0,43	0,32
Apache Xalan	412,72	0	4489	708,01	0,38	0,38
PBeans	268,79	0	3752	684,98	0,59	0,30
Apache Xerces	339,77	0	10701	945,85	0,29	0,21
Apache Ant	295,80	0	4541	417,72	0,51	0,43
Apache Ivy	247,65	1	8095	516,25	0,39	0,31
Apache Camel	111,77	0	2077	160,38	0,34	0,18
Apache Forrest	187,21	1	702	152,81	0,24	0,20
Apache Log4j	177,46	0	2443	220,76	0,35	0,27
Apache Lucene	277,53	1	8474	540,43	0,50	0,37
Apache Poi	288,58	0	9886	603,64	0,29	0,35
Apache Synapse	196,23	0	1449	208,30	0,44	0,36
Apache Velocity	253,31	0	13175	1019,91	0,33	0,29
Ckjm	146,90	1	419	123,45	0,66	0,65
JEdit	457,30	1	23683	1335,81	0,31	0,34
Przemysłowy-1	206,63	0	7313	420,07	0,27	0,17
Przemysłowy-2	162,88	0	18384	388,09	0,24	0,15
Przemysłowy-3	156,15	0	5929	309,24	0,20	0,17
Przemysłowy-4	173,02	0	7114	362,47	0,34	0,19
Przemysłowy-5	127,01	0	5002	210,94	0,28	0,20
Przemysłowy-6	147,83	1	1051	159,72	0,21	0,20
Studencki-1	176,07	3	630	170,91	0,69	0,71
Studencki-2	237,55	2	2451	428,95	0,54	0,31
Studencki-3	497,28	4	1910	619,42	0,56	0,65
Studencki-4	191,45	1	1815	338,35	0,25	0,57

Ciąg dalszy tabeli na następnej stronie.

Tabela A.20: Charakterystyka metryki LOC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Studencki-5	751,63	4	5924	1371,60	0,74	0,81
Studencki-6	56,86	2	348	68,53	0,45	0,37
Studencki-7	105,77	7	861	154,20	0,30	0,32
Studencki-8	315,83	14	992	263,25	0,52	0,48
Studencki-9	336,82	0	2781	360,17	0,04	0,03
Studencki-10	419,33	20	3211	625,41	0,74	0,43
Studencki-11	148,96	4	609	132,65	0,46	0,19
Studencki-12	333,98	1	1541	414,23	0,13	0,32
Studencki-13	480,10	33	2509	559,44	0,43	0,58
Studencki-14	76,40	3	301	70,64	0,20	0,15
Studencki-15	196,17	1	2353	384,07	0,55	0,58
Studencki-16	133,94	0	2090	248,03	0,26	0,19
Studencki-17	233,44	4	1779	374,12	0,47	0,43

Dodatek B

Charakterystyki wartości metryk procesu w projektach

Tabela B.1: Charakterystyka metryki NR

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-1	0,78	0	44	2,28	0,32	0,31
Przemysłowy-2	0,94	0	60	2,22	0,30	0,16
Przemysłowy-3	0,51	0	35	1,40	0,38	0,12
Przemysłowy-4	0,41	0	36	1,37	0,49	0,40
Przemysłowy-5	0,47	0	32	1,17	0,27	-0,02
Przemysłowy-6	0,56	0	6	0,99	0,16	0,14
Apache Xalan	1,98	0	27	1,86	0,35	0,10
Apache Xerces	1,88	0	74	3,89	0,24	0,09
Apache Ant	7,82	0	63	7,14	0,54	0,37
Apache Ivy	6,70	0	120	9,30	0,50	0,35
Apache Camel	1,95	0	60	3,46	0,54	0,52
Apache Log4j	3,22	0	40	5,10	0,49	0,51
Apache Lucene	2,73	0	70	4,72	0,67	0,34
Apache Poi	2,90	0	36	2,98	0,19	-0,03
Apache Synapse	3,17	0	18	3,15	0,40	0,27
Apache Velocity	5,12	0	116	9,74	0,42	0,27
JEdit	6,51	0	169	12,46	0,56	0,36

Tabela B.2: Charakterystyka metryki NDC

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-1	0,52	0	15	1,19	0,33	0,31

Ciąg dalszy tabeli na następnej stronie.

Tabela B.2: Charakterystyka metryki NDC (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-2	0,59	0	16	1,11	0,25	0,16
Przemysłowy-3	0,39	0	13	0,82	0,28	0,12
Przemysłowy-4	0,31	0	9	0,83	0,34	0,40
Przemysłowy-5	0,37	0	9	0,75	0,13	-0,03
Przemysłowy-6	0,41	0	3	0,61	0,12	0,13
Apache Xalan	1,25	0	7	0,87	0,20	0,12
Apache Xerces	1,00	0	7	0,98	0,29	0,11
Apache Ant	3,40	0	10	1,72	0,36	0,30
Apache Ivy	1,41	0	4	0,72	0,32	0,26
Apache Camel	1,03	0	6	1,24	0,41	0,49
Apache Log4j	1,14	0	4	0,87	0,50	0,50
Apache Lucene	1,52	0	8	1,18	0,45	0,31
Apache Poi	1,90	0	9	1,10	0,22	0,02
Apache Synapse	1,57	0	4	1,05	0,26	0,23
Apache Velocity	1,47	0	5	1,21	0,26	0,15
JEdit	1,54	0	16	1,55	-0,08	-0,10

Tabela B.3: Charakterystyka metryki NML

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-1	26,01	0	5584	151,46	0,25	0,31
Przemysłowy-2	20,63	0	4613	104,33	0,26	0,17
Przemysłowy-3	14,73	0	3239	90,53	0,32	0,13
Przemysłowy-4	10,95	0	3661	80,27	0,36	0,38
Przemysłowy-5	18,07	0	2249	83,75	0,24	-0,01

Tabela B.4: Charakterystyka metryki IN

Projekt	Liczba starych klas	Liczba nowych klas	Stare klasy z defektami [%]	Nowe klasy z defaktami [%]
Przemysłowy-1	13730	1916	14,88	0,91
Przemysłowy-2	17550	1866	11,45	0,63
Przemysłowy-3	7429	1035	11,35	0,85
Przemysłowy-4	6273	751	8,94	0,50
Przemysłowy-5	5385	260	18,78	1,36
Przemysłowy-6	349	55	7,43	0,74
Apache Xalan	2316	281	59,18	6,12
Pbeans	21	30	11,76	7,84

Ciąg dalszy tabeli na następnej stronie.

Tabela B.4: Charakterystyka metryki IN (ciąg dalszy)

Projekt	Liczba starych klas	Liczba nowych klas	Stare klasy z defektami [%]	Nowe klasy z defaktami [%]
Apache Xerces	877	604	20,66	18,30
Apache Ant	938	630	14,92	6,12
Apache Ivy	109	484	1,85	7,59
Apache Camel	1688	757	15,34	7,12
Apache Log4j	201	113	40,76	31,21
Apache Lucene	427	160	46,00	13,12
Apache Poi	920	221	44,35	5,26
Apache Synapse	371	107	24,69	5,86
Apache Velocity	364	46	69,27	1,22
JEdit	1071	406	12,53	1,90

Tabela B.5: Charakterystyka metryki NDPV

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-1	0,32	0	37	1,20	0,35	0,32
Przemysłowy-2	0,19	0	27	0,80	0,35	0,22
Przemysłowy-3	0,14	0	8	0,47	0,11	0,03
Przemysłowy-4	0,11	0	9	0,53	0,30	0,15
Przemysłowy-5	0,27	0	19	0,76	0,24	-0,05
Przemysłowy-6	0,49	0	9	0,88	0,40	0,31
Apache Xalan	0,57	0	4	1,06	0,63	0,39
Apache Xerces	0,24	0	11	0,81	0,11	0,10
Apache Ant	0,19	0	10	0,67	0,43	0,29
Apache Ivy	0,95	0	36	3,25	0,55	0,19
Apache Camel	0,35	0	28	1,25	0,41	0,32
Apache Log4j	0,44	0	9	1,17	0,38	0,23
Apache Lucene	1,16	0	47	2,92	0,67	0,34
Apache Poi	0,76	0	20	1,51	0,26	0,09
Apache Synapse	0,25	0	7	0,73	0,37	0,24
Apache Velocity	0,97	0	12	1,90	0,52	0,46
JEdit	0,63	0	45	2,34	0,74	0,45

Tabela B.6: Charakterystyka metryki NER

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-2 (wer. 1)	0,49	0	12	0,98	0,22	0,17
Przemysłowy-2 (wer. 2)	0,35	0	21	1,02	0,38	0,20

Ciąg dalszy tabeli na następnej stronie.

Tabela B.6: Charakterystyka metryki NER (ciąg dalszy)

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-2 (wer. 3)	0,15	0	7	0,48	0,36	0,18
Przemysłowy-2 (wer. 4)	0,31	0	8	0,68	0,27	0,20

Tabela B.7: Charakterystyka metryki NPR

Projekt	Średnia	Minimum	Maksimum	σ	r	r_s
Przemysłowy-2 (wer. 3)	0,05	0	4	0,26	0,31	0,14
Przemysłowy-2 (wer. 4)	0,21	0	7	0,51	0,12	0,04

Spis rysunków

Spis tabel