Politechnika Wrocławska

Wydział Informatyki i Zarządzania
kierunek studiów: Informatyka
specjalność: Inżynieria oprogramowania

Praca dyplomowa - magisterska

Predykcja defektów na poziomie metod w celu zredukowania wysiłku związanego z zapewnieniem jakości oprogramowania

Mateusz Kutyba

słowa kluczowe:
pierwsze
drugie
trzecie

krótkie streszczenie:
Bardzo krótkie streszczenie w którym powinno się znaleźć omówienie tematu pracy i poruszanych terminów. Tekst ten nie może być zbyt długi.

opiekun pracy	dr hab. inż. Lech Madeyski	.....	.....
dyplomowej	*Tytul/stopień naukowy/imię i nazwisko*	*ocena*	*podpis*

*Do celów archiwalnych pracę dyplomową zakwalifikowano do:**
a) kategorii A (akta wieczyste)
b) kategorii BE 50 (po 50 latach podlegające ekspertyzie)
** niepotrzebne skreślić*

pieczątka Instytutu, w którym student wykonywał pracę

Wrocław 2015

Spis treści

Streszczenie

Streszczenie po polsku

Abstract

Abctract in english

Rozdział 1

Wstęp

Prawdopodobnie nie istnieją programy wolne od błędów. Z całą pewnością istnieją programy, które zawierają zbyt dużą liczbę błędów. Każdy kto tworzy oprogramowanie chciałby aby było ono wolne od wad. Podstawowym narzędziem pozwalającym na sprawdzenie czy program działa poprawnie są testy. Powstają coraz bardziej wyszukane metody i metodyki testowania oprogramowania, a wszystko po to oprogramowanie działało zgodnie z oczekiwaniami, czyli aby cechowało się wysoką jakością. Testowanie i inspekcje kodu pozwalają zapewnić odpowiednią jakość oprogramowania, ale są kosztowne. Bazując na zasadzie Pareto [11] wiemy, że około 80% defektów pochodzi z 20% modułów [5]. Wiedząc, które moduły należy poddać inspekcji, można znacznie obniżyć ilość pracy potrzebną do znalezienia większości błędów, a co za tym idzie znacząco zmniejszyć koszt takich inspekcji.

Kod źródłowy oprogramowania zazwyczaj składa się z wielu plików, które są organizowane w pakiety (ang. *packages*). W języku Java pliki zawierają klasy (ang. *class*), a każda klasa może zawierać metody (ang. *method*). Powstało wiele metod predykcji defektów, na różnych poziowach granulacji: od pakietów, przez pliki, klasy, metody, na pojedynczych zmianach (ang. *hunk* [12,13]) skończywszy [4,8,9,19]. Jak wykazano m.in. w [14] i [26], predykcja błędów na poziomie metod dostarcza dokładniejszych danych na temat lokalizacji błędów, dzięki czemu ich odnajdywanie jest efektywniejsze.

Predykcja defektów oprogramowania wykorzystuje techniki eksploracji danych, głównie są to metody statystyczne i metody uczenia maszynowego. Kluczowym elementem w tych procesach są właśnie dane. To na ich podstawie algorytmy uczenia maszynowego są w stanie formułować reguły decyzyjne. W inżynierii oprogramowania tymi danymi są róznego rodzaju metryki oprogramowania. Podział metryk oraz ich zastosowanie zostały szerzej opisane w rozdziale 2.1.

Gromadzenie danych (metryk) z projektów jest czasochłonne, wymaga dużo pracy — pobierania lub kopiowania projektów, mocy obliczeniowej do wyliczenia metryk. Potrzebne jest stworzenie uniwersalnych rozwiązań służących do tego celu oraz nastawienie na możliwość rozszerzania zestawu narzędzi, które mogą być ze sobą dowolnie zestawiane. Te wymagania spełnia platforma DePress [33], która jest rozwijana przy udziale studentów i pracowników Politechniki Wrocławskiej oraz pracowników Capgemini Polska. Więcej informacji o DePress zawarto w rozdziale 3.1.1.

Wstępne przeszukiwanie literatury wykazało niewielką ilość źródeł ściśle odpowiadających zagadnieniu predykcji defektów na niskim poziomie granulacji. Jest to główny

kierunek tych badań a ich celem jest przede wszystkim opracowanie nowego modelu, który miałby służyć do efektywnego wskazywania miejsc w oprogramowaniu, w których znajdują się błędy. Pozwololiłoby to na ograniczenie ilości pracy potrzebnej do przejrzenia krytycznych miejsc i naprawienia błędów.

1.1. Cele pracy

Cele pracy dyplomowej:

- Przegląd literatury pod kątem predykcji defektów oprogramowania, szczególnie na niskim poziomie granulacji.
- Budowa nowych lub rozbudowa istniejących narzędzi służących do wyliczenia metryk oprogramowania, współpracujących z wersjonowanymi repozytoriami kodu (SVN oraz Git).
- Zebranie danych z projektów o otwartych źródłach na potrzeby predykcji defektów.
- Budowa modeli predykcji z wykorzystaniem zebranych danych.
- Ocena stworzonych narzędzi oraz zebranych danych.
- Ewaluacja modeli predykcji i ocena ich skuteczności.

1.2. Związek z innymi pracami

Na początku prac dokonano przeglądu literatury aby określić aktualny stan wiedzy (ang. *state of the art*) w badanej dziedzinie. Przegląd literatury pozwolił udzielić odpowiedzi na następujące pytania:

- **Jakie istnieją metody predykcji defektów na poziomie metod i jaka jest ich skuteczność?**
  Istnieje wiele modeli predykcji defektów, jednak większość z nich opiera się na danych dotyczących klas, pakietów lub modułów. Odpowiedzią na powyższe pytanie jest zbiór modeli predykcji defektów na niskim poziomie granulacji, na przykład metod lub bloków kodu.
- **Jakie są możliwości usprawnienia lub rozwinięcia istniejących metod?**
  Zostały zebrane wszelkie możliwości ulepszenia lub rozszerzenia badań wskazanych przez autorów, określone np. jako "Dalszy rozwój".
- **Jakie są sposoby ekstrakcji zmian kodu źródłowego na poziomie metod?**
  Jakie są sposoby porównywania wersji kodu źródłowego, jakiego rodzaju dane (metryki) są uzyskiwane.

Podczas wstępnego rozpoznania dziedziny zauważono, że liczba publikacji jest niewielka. W związku z tym postanowiono przeprowadzić wyszukiwanie w dwóch etapach. W pierwszym etapie przeszukano elektroniczne zbiory, natomiast w drugim etapie przejrzano bibliografie pozyskanych publikacji a także wszystkie publikacje ich autorów w celu odnalezienia dodatkowych tekstów.

Przeszukiwalne zbiory cyfrowe. Przeszukano poniższe zbiory z użyciem ustalonych wyrażeń, za pomocą wyszukiwarek udostępnianych w postaci aplikacji internetowej:

- IEEE Xplore,
- Science Direct,
- ACM Digital Library,
- Springer Link,
- ISI Web of Science.

Wybrano te zbiory ponieważ pokrywają one większość publikacji inżynierii oprogramowania oraz są używane jako źródła w innych przeglądach z tej dziedziny [19, 48].

Szara literatura. Wstępne przeszukiwanie wykazało niewielką ilość źródeł ściśle odpowiadających zagadnieniu predykcji defektów na niskim poziomie granulacji. Aby pokryć znaczną część szarej literatury zdecydowano, aby przeszukać następujące źródła:

- Google Scholar.
- Lista odnośników w znalezionych źródłach pierwotnych.
  Zgodnie z metodą śnieżnej kuli [17] przejrzano listy referencji źródeł pierwotnych w celu odnalezienia dodatkowych istotnych (relewantnych) publikacji.
- Inne publikacje autorów znalezionych źródeł pierwotnych.
  Przeszukano bazę DBLP [31] szukając według nazwisk autorów dotychczas zgromadzonych źródeł pierwotnych.

Po dokonaniu przeglądu literatury oraz oceny znalezionych źródeł, wybrano te najbardziej istotne z punktu widzenia niniejszej pracy:

- Declarative visitors to ease fine-grained source code mining with full history on billions of AST nodes. [10]
- Method-level bug prediction. [14]
- Comparing fine-grained source code changes and code churn for bug prediction. [15]
- Fault-prone Module Prediction Using Version Histories. [23]
- Reconstructing fine-grained versioning repositories with git for method-level bug prediction. [24]
- Historage: fine-grained version control system for Java. [25]
- Bug prediction based on fine-grained module histories. [26]

1.3. Struktura pracy

TODO Omówienie zawartości rozdziałów

Rozdział 2

Wprowadzenie

2.1. Rola metryk w inżynierii oprogramowania

zastosowania metryk do różnych celów

krótko o rodzajach metryk, do czego można ich użyć

- metryki procesu dają lepsze efekty
- HCM?? [20]
- metryki organizacji nie korelują z błędami [26]
- ważność historycznych metryk [3, 7, 18, 20, 22, 27, 28, 35, 38–41, 44, 46, 50, 51]

2.2. Systemy kontroli wersji jako źródło danych o projektach

jak się wyciąga historie metod (fine-grained metrics)

- C-REX [21]
- BEAGLE [16]
- Kenyon [2]
- APFEL [53]
- Historage [26]
- ChangeDistiller [14]

2.3. Wyciąganie i linkowanie bugów (przeredagować tytuł)

ogólnie o ITS: jira, bugzilla, IBM, ...

2.3.1. metody

- SZZ [49]
- ReLink [52]
- MLink [42]
- HIPIKAT [6]

2.4. Sformułowanie problemu

Głównym celem badań jest stworzenie narzędzi w postaci wtyczek do środowiska KNIME, służących do gromadzenia metryk oprogramowania z systemów kontroli wersji oraz zgromadzenie jak największej ilości metryk w publicznym repozytorium.

Następnym krokiem będzie stworzenie modelu (modeli) predykcji defektów oraz ich ewaluacja.

Stworzenie narzędzi pozwalających na zautomatyzowane gromadzenie metryk z dostępnych projektów (na przykład Open Source) pozwoli rozszerzyć publiczne zbiory danych o bardzo dużą liczbę metryk.

Powstanie dzięki temu możliwość tworzenia skuteczniejszych modeli predykcji defektów oprogramowania dzięki: większym zbiorom uczącym; ewaluacji modeli na większych zbiorach danych.

Napisać coś więcej pod kątem zapewnienia jakości, zredukowania wysiłku.

2.4.1. Koszty zapewnienia jakości

Koszty zapewnienia jakości są prawie proporcjonalne do wielkości modułu [1].

File-level są bardziej efektywne niż package-level [27, 43, 45].

Średnio plik ma 10 metod a błędy są tylko w 1–2 metodach.

Method-level [28, 37]

Zmniejszyć koszt, wysiłek to obecnie główny kierunek [1, 29, 34, 36, 47].

2.4.2. Przeznaczenie narzędzi

Przeznaczenie narzędzi, DePress platforma (framework)

2.4.3. Wymagania

Wymagania

2.4.4. Ograniczenia dotyczące realizacji

Ograniczenia dotyczące realizacji (narzucone, założone)

Stworzenie wtyczek do Knime, służących do wyliczania metryk procesu na podstawie danych z systemów kontroli wersji, zgromadzenie metryk z projektów Open Source w repozytorium.

Założenia: wtyczki w środowisku Knime, wykorzystanie istniejących rozwiązań platformy DePress, systemy kontroli wersji: git, svn.

Ograniczenia zakresu tematycznego: badanie tylko projektów pisanych w Java

2.4.5. Możliwości realizacji zadań

Możliwości realizacji zadań - wstępna dyskusja

2.4.6. Metoda oceny

Effort-based evaluation (na polski)

Poddajemy inspekcji 20% LOC albo 20% instrukcji (bo 1 LOC $\neq$ 1 instrukcja) lub poddajemy inspekcji określoną liczbę LOC (stały próg, bo 20% LOC z dużego projektu może być zbyt wiele) — ile % błędów znaleziono

Rozdział 3

Omówienie infrastruktury pomiarowej

...

3.1. Knime

Knime

3.1.1. DePress

3.2. Weka

Weka

3.3. AstCompare (roboczo)

AstCompare plugin

3.3.1. Wymagania

3.3.2. Implementacja

3.3.3. Testowanie i gromadzenie danych

Metryki zapoczątkowane przez autorów i zaimplementowane przez poprzednika (Piotr Mitka [**?**]):

- allMethodHistories
- methodHistories
- authors
- stmtAdded
- maxStmtAdded
- avgStmtAdded
- stmtUpdated
- maxsSmtUpdated
- avgStmtUpdated
- stmtDeleted
- maxStmtDeleted
- avgStmtDeleted
- stmtParentChanged
- churn
- maxChurn

- avgChurn
- decl
- cond

Metryki dodane przeze mnie:

- elseAdded
- elseDeleted
- loopsAdded
- loopsUpdated
- loopsDeleted
- variablesAdded
- variablesUpdated
- variablesDeleted
- assigmentsAdded
- assigmentsUpdated
- assigmentsDeleted
- returnsAdded
- returnsUpdated
- returnsDeleted
- nullsAdded
- nullsUpdated
- nullsDeleted
- casesAdded
- casesUpdated
- casesDeleted
- breaksAdded
- breaksUpdated
- breaksDeleted
- objectsAdded
- objectsUpdated
- objectsDeleted
- catchesAdded
- catchesUpdated
- catchesDeleted
- throwsAdded
- throwsUpdated
- throwsDeleted

Dodatkowo zaimplementowano drugie wyjście wtyczki z wszystkimi historiami metod (do linkowania błędów): commit_id, nazwa metody, autor, komentarz, data.

3.3.4. Omówienie zebranych danych pomiarowych

Wymagania dla repozytorium metryk

Struktura danych i zawartość zbioru

Rozdział 4

Modele predykcji i ich ewaluacja

Wykorzystanie RandomForest [26, 27, 30, 32, 34]

Udział % metod z błędami jest niewielki — modele predykcji mają tendencję do wskazywania wszystkich jako 0, bo mało FP — logistic regression tak miało w [26].

Walidacja modelu przez 10-fold cross validation oraz przez testowanie na zbiorze danych z następnego wydania (ang. *release*).

Wykorzystywane algorytmy:

- RandomForest
- J48
- NaiveBayes
- Bagging
- AdaBoostM1
- RacedIncrementalLogitBoost
- LMT
- BFTree
- RandomTree

Przebadane projekty:

Rys. 4.1. Projekty

robocza tabelka							
			NCSS	# of methods	Buggy methods	% of buggy	daty
log4j	v_1_2final	v1_2_11	9904	1746	70	4,01%	...
ecf	Root_Release_3_0	Root_Release_3_3	63375	12260	174	1,42%	...
jdt.debug	R2_0	R3_0	103466	9652	172	1,78%	...
ant	ANT_170	ANT_180	56487	9697	296	3,05%	
cassandra	cassandra-1.2.0-rc2	cassandra-2.0.0	93654	15545	557	3,58%	

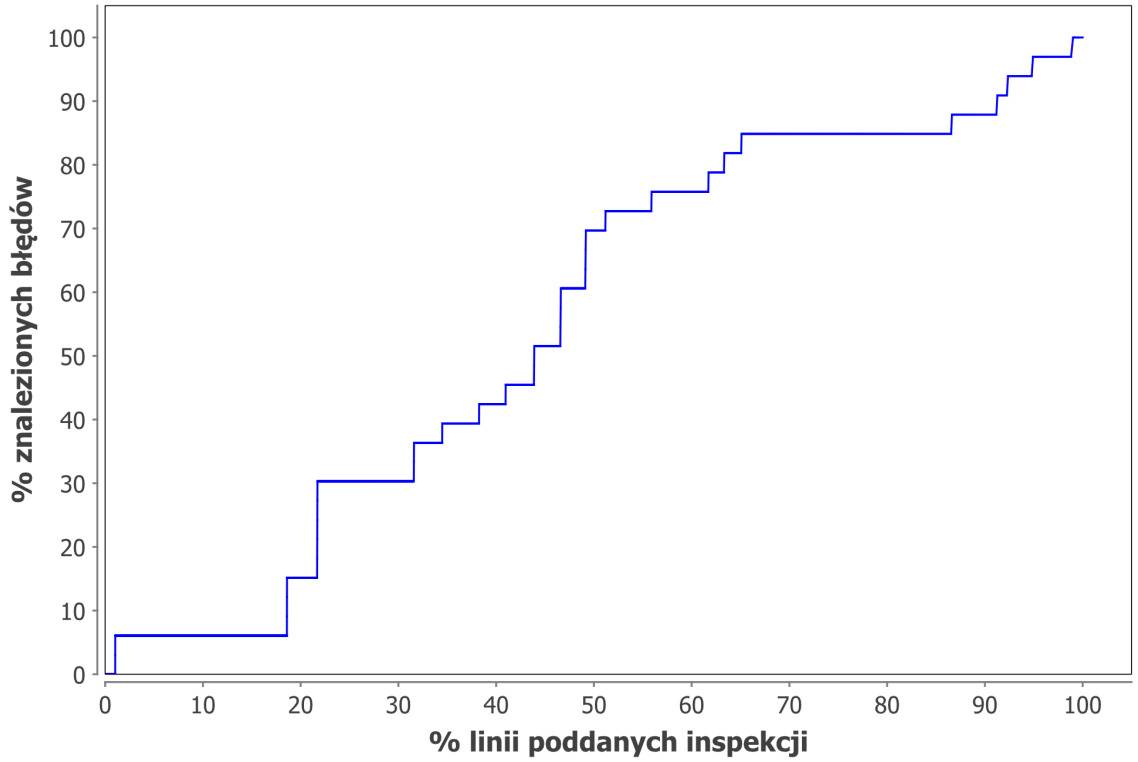
- NCSS = Non Commenting Source Statements
- pierwszy wiersz w wyniku (Sort by NCSS) polega na posortowaniu metod od najmniejszych i sprawdzaniu ich w tej kolejności
- kolejne wyniki są dla modeli opartych na wymienionych wyżej algorytmach klasyfikacji
- wyniki porównuję głównie na podstawie efektywności znajdowania błędów — sortuję wyniki wg prawdopodobieństwa wystąpienia błędu w metodzie (algorytmy

Rys. 4.2. Wyniki

		% of bugs found in 20% of NCSS	Accuracy	Kappa	AUC
log4j	Sort by NCSS	9,68	?	?	?
	RandomForest	92,86	0,99	0,81	0,94
	J48	82,86	0,98	0,64	0,76
	NaiveBayes	71,43	0,96	0,44	0,75
	Bagging	84,29	0,98	0,57	0,89
	AdaBoostM1	75,71	0,97	0,51	0,77
	RacedIncrementalLogitBoost	77,14	0,96	0,00	0,51
	LMT	78,57	0,97	0,47	0,80
	BFTree	80,00	0,97	0,58	0,78
	RandomTree	90,00	0,99	0,83	0,92
ecf	Sort by NCSS	25,00	?	?	?
	RandomForest	81,03	0,99	0,56	0,84
	J48	55,17	0,99	0,12	0,54
	NaiveBayes	50,57	0,97	0,20	0,72
	Bagging	59,20	0,99	0,10	0,77
	AdaBoostM1	62,07	0,99	0,00	0,73
	RacedIncrementalLogitBoost	58,62	0,98	0,10	0,67
	LMT	51,72	0,99	0,00	0,50
	BFTree	51,72	0,99	0,00	0,50
	RandomTree	77,59	0,99	0,59	0,76
jdt.debug	Sort by NCSS	64,53	?	?	?
	RandomForest	100,00	0,98	0,52	0,84
	J48	100,00	0,98	0,47	0,75
	NaiveBayes	100,00	0,98	0,36	0,75
	Bagging	100,00	0,98	0,43	0,80
	AdaBoostM1	100,00	0,97	0,42	0,78
	RacedIncrementalLogitBoost	100,00	0,98	0,00	0,50
	LMT	100,00	0,98	0,43	0,76
	BFTree	100,00	0,98	0,49	0,76
	RandomTree	100,00	0,98	0,48	0,73
ant	Sort by NCSS	15,15	?	?	?
	RandomForest	57,43	0,97	0,44	0,84
	J48	47,30	0,97	0,33	0,70
	NaiveBayes	31,76	0,94	0,24	0,76
	Bagging	40,54	0,97	0,29	0,81
	AdaBoostM1	34,12	0,97	0,00	0,76
	RacedIncrementalLogitBoost	33,11	0,96	0,25	0,74
	LMT	33,45	0,97	0,21	0,78
	BFTree	36,82	0,97	0,22	0,67
	RandomTree	57,77	0,97	0,45	0,71
cassandra	Sort by NCSS	0,00	?	?	?
	RandomForest	83,84	0,96	0,41	0,74
	J48	82,94	0,95	0,32	0,61
	NaiveBayes	68,04	0,96	0,23	0,64
	Bagging	74,33	0,96	0,35	0,70
	AdaBoostM1	79,53	0,95	0,31	0,65
	RacedIncrementalLogitBoost	79,53	0,95	0,29	0,62
	LMT	75,04	0,95	0,33	0,66
	BFTree	85,28	0,95	0,34	0,64
	RandomTree	91,74	0,96	0,41	0,67

podają zwycięzcę oraz prawdopodobieństwa dla 0 i 1), następnie zliczam ile % błędów zostanie odnalezione przy przeglądaniu 20% kodu w tej kolejności
- wyniki w tabeli poniżej
- przykładowe wykresy efektywności dla 2 projektów poniżej
- na wykresach wybrane 2 projekty które wyszły najgorzej, w pozostałych projektach te krzywe są jeszcze bardziej strome

Rys. 4.3. Ant — Sort by NCSS

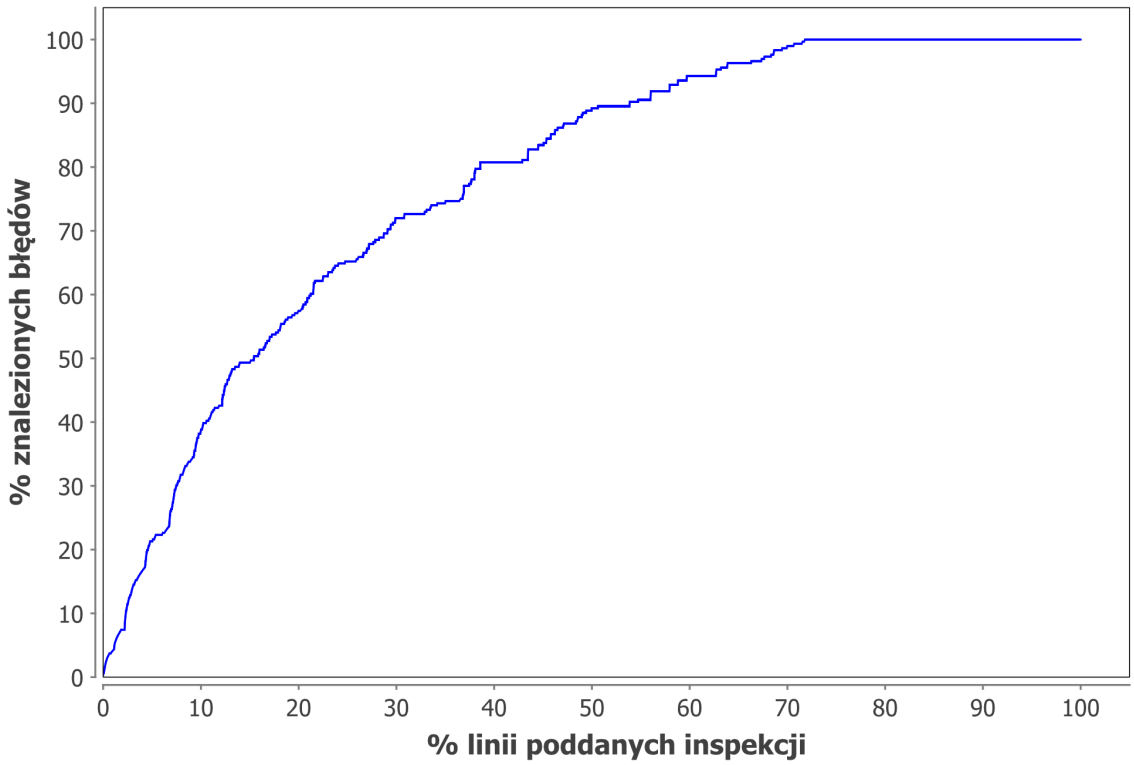


4.1. Ocena rozwiązania

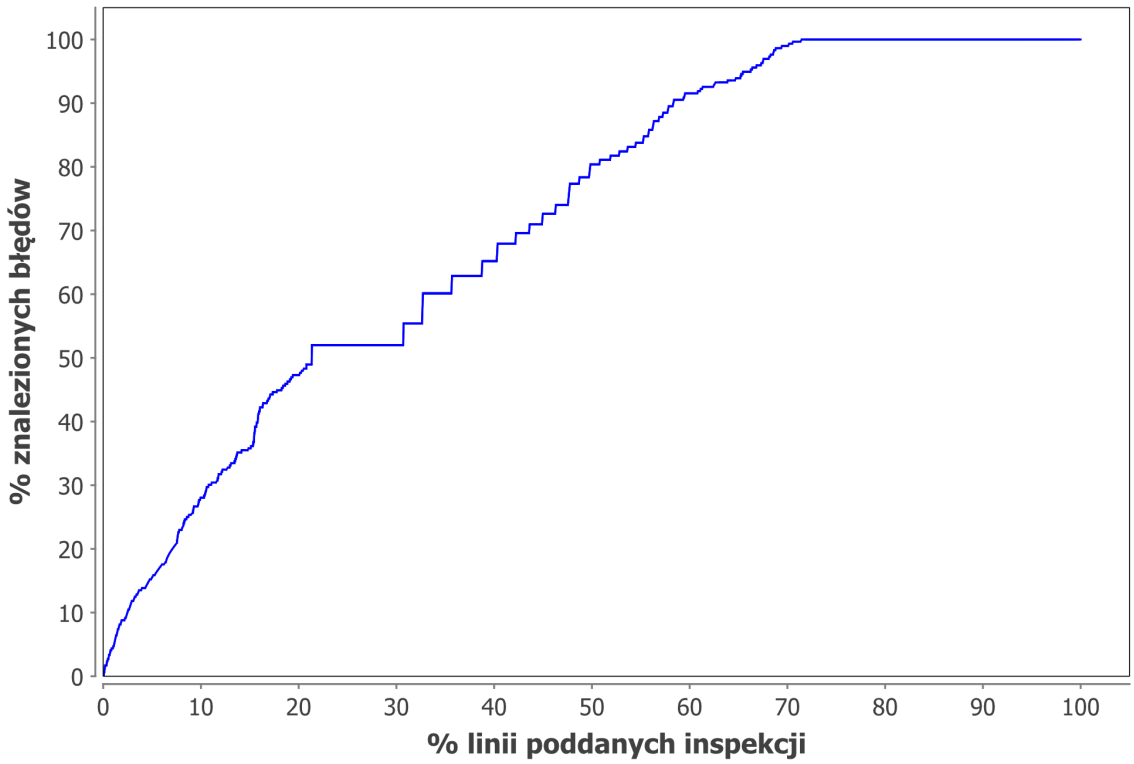
np. Goal-Question-Metric

- Stopień realizacji wymagań funkcjonalnych
- Poprawność rozwiązania (funkcjonowania systemu): weryfikacja (symulacja), testowanie
  utworzenie nowego modelu predykcji defektów, bazującego na zgromadzonych metrykach w celu weryfikacji przydatności narzędzi i zgromadzonych metryk
- Właściwości (parametry) rozwiązania
- Porównanie z innymi rozwiązaniami

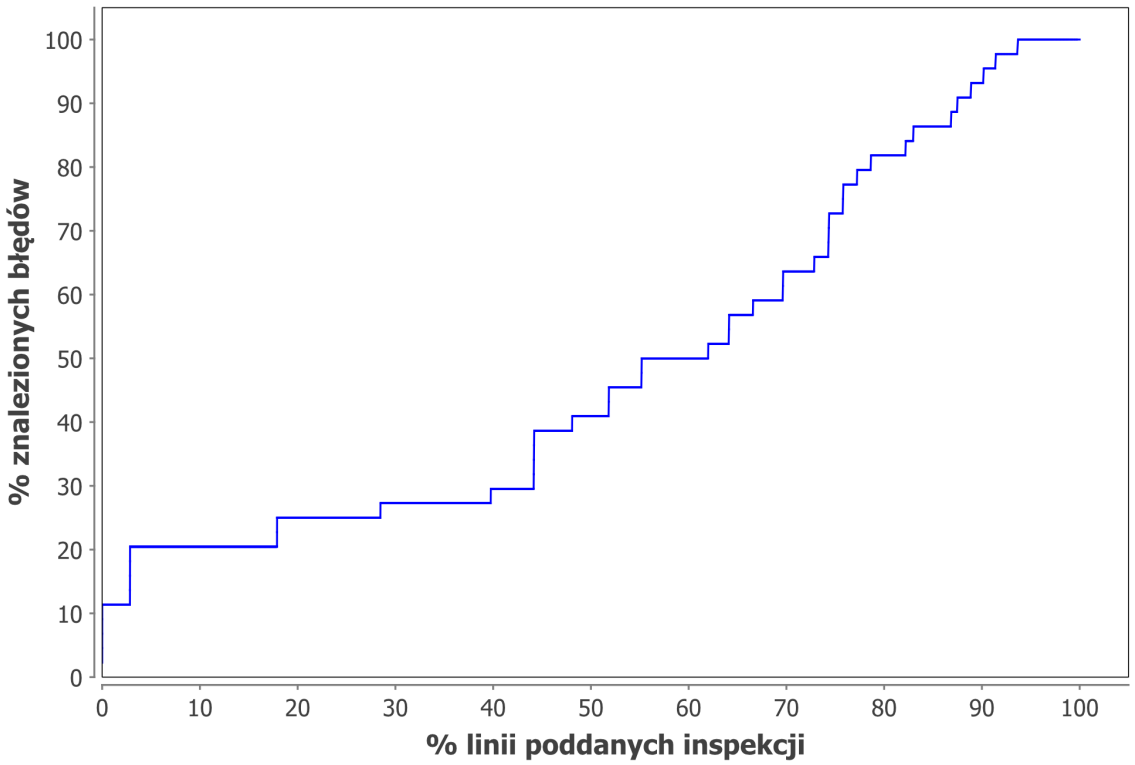
Rys. 4.4. Ant — RandomForest



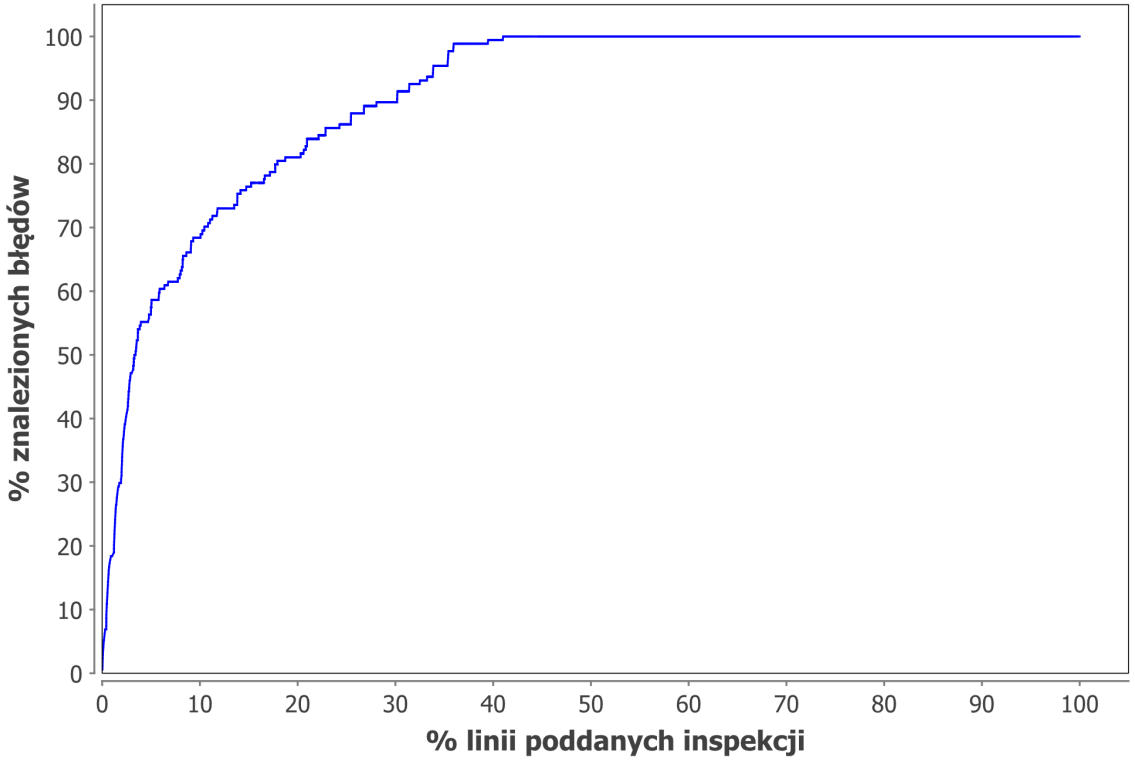
Rys. 4.5. Ant — J48



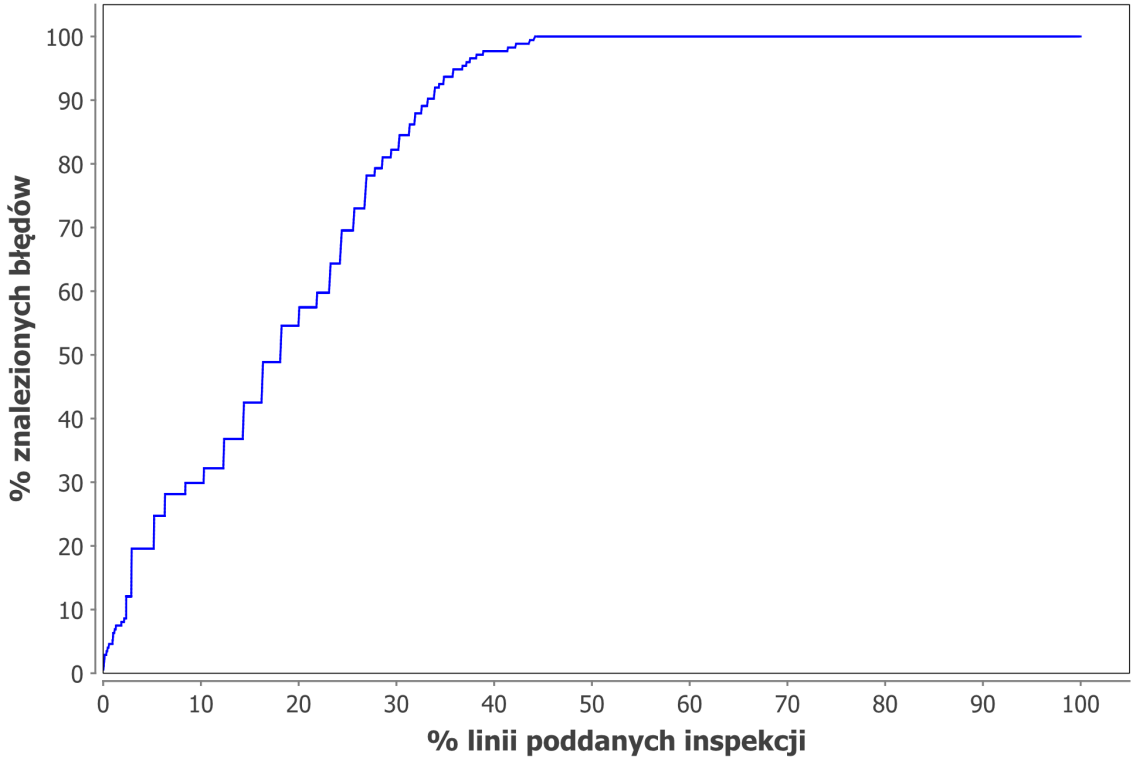
Rys. 4.6. ECF — Sort by NCSS



Rys. 4.7. ECF — RandomForest



Rys. 4.8. ECF — J48



Rozdział 5

Podsumowanie i propozycja dalszych badań

- Wady, zalety, ograniczenia
- Wnioski
- Zakres zastosowań, rozwój i dalsze badania

Spis ilustracji

Spis tabel

Bibliografia

[1] Arisholm E., Briand L. C., Johannessen E. B. A systematic and comprehensive investigation of methods to build and evaluate fault prediction models. *Journal of Systems and Software*, 83(1):2–17, 2010.

[2] Bevan J., Whitehead Jr E. J., Kim S., Godfrey M. Facilitating software evolution research with kenyon. In *ACM SIGSOFT Software Engineering Notes*, volume 30, pages 177–186. ACM, 2005.

[3] Bird C., Nagappan N., Murphy B., Gall H., Devanbu P. Don't touch my code!: examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 4–14. ACM, 2011.

[4] Catal C., Diri B. A systematic review of software fault prediction studies. *Expert systems with applications*, 36(4):7346–7354, 2009.

[5] Concas G., Marchesi M., Murgia A., Tonelli R., Turnu I. On the distribution of bugs in the eclipse system. *Software Engineering, IEEE Transactions on*, 37(6):872–877, 2011.

[6] Cubranic D., Murphy G. C. Hipikat: Recommending pertinent software development artifacts. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 408–418. IEEE, 2003.

[7] Czerwonka J., Das R., Nagappan N., Tarvo A., Teterev A. Crane: Failure prediction, change analysis and test prioritization in practice–experiences from windows. In *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on*, pages 357–366. IEEE, 2011.

[8] D'Ambros M., Lanza M., Robbes R. An extensive comparison of bug prediction approaches. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 31–41. IEEE, 2010.

[9] D'Ambros M., Lanza M., Robbes R. Evaluating defect prediction approaches: a benchmark and an extensive comparison. *Empirical Software Engineering*, 17(4-5):531–577, 2012.

[10] Dyer R., Rajan H., Nguyen T. N. Declarative visitors to ease fine-grained source code mining with full history on billions of ast nodes. In *Proceedings of the 12th international conference on Generative programming: concepts & experiences*, pages 23–32. ACM, 2013.

[11] Endres A., Rombach H. D. *A handbook of software and systems engineering: empirical observations, laws and theories*. Pearson Education, 2003.

[12] Ferzund J., Ahsan S. N., Wotawa F. Empirical evaluation of hunk metrics as bug predictors. In *Software Process and Product Measurement*, pages 242–254. Springer, 2009.

[13] Ferzund J., Ahsan S. N., Wotawa F. Software change classification using hunk metrics. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 471–474. IEEE, 2009.

[14] Giger E., D'Ambros M., Pinzger M., Gall H. C. Method-level bug prediction. In *Proceedings of the ACM-IEEE international symposium on Empirical software engineering and measurement*, pages 171–180. ACM, 2012.

[15] Giger E., Pinzger M., Gall H. C. Comparing fine-grained source code changes and code churn for bug prediction. In *Proceedings of the 8th Working Conference on Mining Software Repositories*, pages 83–92. ACM, 2011.

[16] Godfrey M. W., Zou L. Using origin analysis to detect merging and splitting of source code entities. *Software Engineering, IEEE Transactions on*, 31(2):166–181, 2005.

[17] Goodman L. A. Snowball sampling. *The annals of mathematical statistics*, pages 148–170, 1961.

[18] Graves T. L., Karr A. F., Marron J. S., Siy H. Predicting fault incidence using software change history. *Software Engineering, IEEE Transactions on*, 26(7):653–661, 2000.

[19] Hall T., Beecham S., Bowes D., Gray D., Counsell S. A systematic literature review on fault prediction performance in software engineering. *Software Engineering, IEEE Transactions on*, 38(6):1276–1304, 2012.

[20] Hassan A. E. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.

[21] Hassan A. E., Holt R. C. C-rex: an evolutionary code extractor for c. In *CSER Meeting*. Citeseer, 2004.

[22] Hassan A. E., Holt R. C. The top ten list: Dynamic fault prediction. In *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 263–272. IEEE, 2005.

[23] Hata H. Fault-prone module prediction using version histories. 2012.

[24] Hata H., Mizuno O., Kikuno T. Reconstructing fine-grained versioning repositories with git for method-level bug prediction. *IWESEP '10*, pages 27–32, 2010.

[25] Hata H., Mizuno O., Kikuno T. Historage: fine-grained version control system for java. In *Proceedings of the 12th International Workshop on Principles of Software Evolution and the 7th annual ERCIM Workshop on Software Evolution*, pages 96–100. ACM, 2011.

[26] Hata H., Mizuno O., Kikuno T. Bug prediction based on fine-grained module histories. In *Proceedings of the 2012 International Conference on Software Engineering*, pages 200–210. IEEE Press, 2012.

[27] Kamei Y., Matsumoto S., Monden A., Matsumoto K.-i., Adams B., Hassan A. E. Revisiting common bug prediction findings using effort-aware models. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[28] Kim S., Zimmermann T., Whitehead Jr E. J., Zeller A. Predicting faults from cached history. In *Proceedings of the 29th international conference on Software Engineering*, pages 489–498. IEEE Computer Society, 2007.

[29] Koru A. G., El Emam K., Zhang D., Liu H., Mathew D. Theory of relative defect proneness. *Empirical Software Engineering*, 13(5):473–498, 2008.

[30] Lessmann S., Baesens B., Mues C., Pietsch S. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, 34(4):485–496, 2008.

[31] Ley M., Herbstritt M., Ackermann M. R., Hoffmann O., Wagner M., von Keutz S., Hostert K. Dblp bibliography, 2014.

[32] Liaw A., Wiener M. Classification and regression by randomforest. *R news*, 2(3):18–22, 2002.

[33] Madeyski L., Majchrzak M. Software measurement and defect prediction with depress extensible framework. *Foundations of Computing and Decision Sciences*, 39(4):249–270, 2014.

[34] Mende T., Koschke R. Effort-aware defect prediction models. In *Software Maintenance and Reengineering (CSMR), 2010 14th European Conference on*, pages 107–116. IEEE, 2010.

[35] Meneely A., Williams L., Snipes W., Osborne J. Predicting failures with developer networks and social network analysis. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 13–23. ACM, 2008.

[36] Menzies T., Milton Z., Turhan B., Cukic B., Jiang Y., Bener A. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering*, 17(4):375–407, 2010.

[37] Mizuno O., Kikuno T. Training on errors experiment to detect fault-prone software modules by spam filter. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 405–414. ACM, 2007.

[38] Mockus A. Organizational volatility and its effects on software defects. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, pages 117–126. ACM, 2010.

[39] Moser R., Pedrycz W., Succi G. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 181–190. IEEE, 2008.

[40] Nagappan N., Ball T. Use of relative code churn measures to predict system defect density. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pages 284–292. IEEE, 2005.

[41] Nagappan N., Murphy B., Basili V. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering*, pages 521–530. ACM, 2008.

[42] Nguyen A. T., Nguyen T. T., Nguyen H. A., Nguyen T. N. Multi-layered approach for recovering links between bug reports and fixes. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, page 63. ACM, 2012.

[43] Nguyen T. H., Adams B., Hassan A. E. Studying the impact of dependency network measures on software quality. In *Software Maintenance (ICSM), 2010 IEEE International Conference on*, pages 1–10. IEEE, 2010.

[44] Pinzger M., Nagappan N., Murphy B. Can developer-module networks predict failures? In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*, pages 2–12. ACM, 2008.

[45] Posnett D., Filkov V., Devanbu P. Ecological inference in empirical software engineering. In *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, pages 362–371. IEEE Computer Society, 2011.

[46] Rahman F., Devanbu P. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering*, pages 491–500. ACM, 2011.

[47] Rahman F., Posnett D., Hindle A., Barr E., Devanbu P. Bugcache for inspections: hit or miss? In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 322–331. ACM, 2011.

[48] Riaz M., Mendes E., Tempero E. A systematic review of software maintainability prediction and metrics. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 367–377. IEEE Computer Society, 2009.

[49] Śliwerski J., Zimmermann T., Zeller A. When do changes induce fixes? *ACM sigsoft software engineering notes*, 30(4):1–5, 2005.

[50] Weyuker E. J., Ostrand T. J., Bell R. M. 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.

[51] Wolf T., Schroter A., Damian D., Nguyen T. Predicting build failures using social network analysis on developer communication. In *Proceedings of the 31st International Conference on Software Engineering*, pages 1–11. IEEE Computer Society, 2009.

[52] Wu R., Zhang H., Kim S., Cheung S.-C. Relink: recovering links between bugs and changes. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, pages 15–25. ACM, 2011.

[53] Zimmermann T. Fine-grained processing of cvs archives with apfel. In *Proceedings of the 2006 OOPSLA workshop on eclipse technology eXchange*, pages 16–20. ACM, 2006.

Dodatek A

Coś dodatkowego