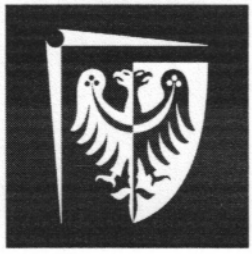

Politechnika Wrocławska

Wrocław, 18 września 2020

dr hab. inż. Lech Madeyski, prof. uczelni
Katedra Informatyki Stosowanej
Wydział Informatyki i Zarządzania
Politechnika Wrocławska
✉: lech.madeyski@pwr.edu.pl
🏠: http://madeyski.e-informatyka.pl

Recenzja rozprawy doktorskiej mgra Mikołaja Fejzera pt. "Mining software repositories for code quality"

1 Wstęp

Niniejsza recenzja została opracowana w odpowiedzi na list prof. dra hab. Pawła Strzeleckiego, Dziekana Wydziału Matematyki, Informatyki i Mechaniki Uniwersytetu Warszawskiego, z dnia 13 lipca 2020 roku, z prośbą o ocenę rozprawy doktorskiej Pana mgra Mikołaja Fejzera nt.: "*Mining software repositories for code quality (Eksploracja repozytoriów kodu w celu zapewnienia jakości oprogramowania)*".

2 Tematyka rozprawy

Tematyka rozprawy mgra Mikołaja Fejzera obejmuje kilka, być może nieco arbitralnie wybranych przez Autora, zagadnień w postaci problemów mających realne, praktyczne znaczenie w inżynierii oprogramowania. Część z nich ma charakter badań wstępnych, co prawdopodobnie wiąże się z etapem poszukiwania tematów, na których Doktorant mógłby w pełni skupić swoją uwagę.

Dotyczą one obserwacji zachowań współpracowników (kontrybutorów) w projektach o otwartych źródłach (ang. *open source*) z zamiarem zdobycia wglądu w role grup programistów, a także możliwości automatycznej detekcji błędów. Następnie Autor skupia się na dwóch kluczowych i moim zdaniem aktualnych zagadnieniach badawczych. Pierwszy to problem rekomendacji recenzentów na potrzeby przeglądu kodu, drugi to problem lokalizacji błędów w kodzie źródłowym na podstawie zgłoszeń błędów. Choć nie wszystkie poruszane zagadnienia są ze sobą bardzo mocno powiązane, to ich rozwiązanie zawsze wymaga eksploracji repozytoriów oprogramowania (ang. *Mining Software Repositories* (MSR)) z wykorzystaniem metod wyszukiwania informacji i uczenia maszynowego, co stanowi swego rodzaju klamrę spinającą doktorat. To pozwala zakwalifikować pracę do niezwykle dynamicznie rozwijającego się w ostatnich latach nurtu MSR w ramach inżynierii oprogramowania. Warto zauważyć, że flagowa konferencja MSR, reprezentująca nurt o tej samej nazwie, jest klasyfikowana bardzo wysoko (na poziomie A) w rankingu CORE. Podsumowując tematykę pracy Doktoranta uważam za aktualną.

3 Treść rozprawy

Rozprawa napisana jest w języku angielskim, składa się z siedmiu rozdziałów (przy czym rozdział o numerze 7 to w istocie załącznik), bibliografii, i liczy 91 stron.

Rozdział 1 wprowadza w tematykę rozprawy. Prezentuje motywacje (w postaci chęci zapewnienia jakości tworzonego oprogramowania poprzez eksplorację repozytoriów oprogramowania z wykorzystaniem technik wyszukiwania informacji i uczenia maszynowego) oraz wskazuje kluczowe problemy badawcze (w szczególności rekomendację recenzentów na potrzeby przeglądu kodu oraz lokalizację błędów w kodzie źródłowym na podstawie zgłoszeń błędów).

Rozdział 2 prezentuje i formalizuje terminologię i podstawowe pojęcia z obszaru inżynierii oprogramowania, wyszukiwania informacji, uczenia maszynowego oraz eksploracji repozytoriów oprogramowania. Autor wykorzystuje je w dalszej części pracy. Są one na ogół wystarczająco precyzyjne, choć pojawiają się drobne wątpliwości.

W rozdziale 3, Autor (bazując na dwóch publikacjach konferencyjnych, których jest pierwszym Autorem [1, 2]) przedstawia analizę projektów o otwartych źródłach (ang. *open source*) skupiając się na dwóch zagadnieniach. Pierwszym jest obserwacja zachowań współpracowników w 42 projektach i wskazanie grup programistów odpowiedzialnych za większość wykonanej pracy. Zaprezentowana metoda jest w stanie wykryć głównych współpracowników dla analizowanych projektów. Jednocześnie analiza komentarzy pokazuje, że większość współpracowników nie angażuje się w komentowanie zestawów zmian. Drugim poruszanym zagadnieniem jest przygotowanie klasyfikatora wykrywającego błędy podczas inspekcji kodu w wersjonowanych repozytoriach *git* i jego empiryczna ewaluacja na 64 projektach, o różnych długościach historii zmian oraz wykorzystujących różne języki programowania. Badania przedstawione w rozdziale 3 mają charakter wstępny i służą niejako "wejściu" w kluczowe tematy, na których Autor skupia swoją uwagę w kolejnych dwóch rozdziałach.

Rozdział 4 bazuje na artykule w czasopiśmie *Journal of Intelligent Information Systems* z listy JCR o współczynniku wpływu (ang. *impact factor*) w roku publikacji wynoszącym 1.589, którego Doktorant jest pierwszym autorem [3]. Rozdział przedstawia pierwszy z dwóch głównych problemów badawczych, na których skupia się Autor w rozprawie doktorskiej i dotyczy rekomendowania recenzentów na etapie przeglądu propozycji zmian kodu źródłowego (ang. *code*

review), przed ich ewentualnym zaakceptowaniem. Formalne inspekcje kodu, a następnie lekkie przeglądy kodu mają długą historię, a te ostatnie są powszechnie stosowane w wielu współczesnych projektach inżynierii oprogramowania. Wybór recenzentów zmian kodu jest realnym problemem z którym spotykają się inżynierowie oprogramowania, w szczególności w dużych, rozproszonych projektach, a sama tematyka automatyzacji wyboru recenzentów cieszy się również sporym zainteresowaniem badaczy i praktyków. W związku z tym może zostać ona uznana za trafnie wybraną i ważną z praktycznego punktu widzenia. Autor przedstawił metodę automatycznego doboru recenzentów na potrzeby przeglądu kodu, przeprowadził empiryczną ewaluację i porównanie tej metody z niektórymi wcześniej stosowanymi metodami.

Rozdział 5 dotyczy drugiego, kluczowego problemu badawczego poruszanego w rozprawie doktorskiej, a mianowicie adaptacyjnej metody lokalizacji błędów programistycznych na podstawie zgłoszeń tychże błędów. Problem lokalizacji błędów jest bardzo ważny z praktycznego punktu widzenia. Zastosowanie skutecznej metody pozwala bowiem sprawniej i taniej naprawiać zgłoszone błędy. Jest to też obszar aktywnych badań naukowych. Problem lokalizacji błędów Autor formułuje jako problem wyszukiwania informacji, gdzie raporty o błędach traktujemy jako zapytania, a kod źródłowy jako zbiór dokumentów. W efekcie pliki kodu źródłowego są wybierane na podstawie ich zgodności z konkretnym raportem. Rozdział ten stanowi moim zdaniem najdojrzalszą część pracy, jedyną, która zawiera cenną dyskusję zagrożeń dla wiarygodności badań. Jeśli założyć, że kolejność rozdziałów wynika z chronologii prowadzonych badań, to w porównaniu z rozdziałem 3, widać postęp jaki poczynił Doktorant w trakcie rozwiązywania kolejnych problemów badawczych. Zgodnie z deklaracją Autora, rozdział ten bazuje na artykule, który jest w trakcie recenzji.

Rozdział 6 zawiera krótkie podsumowanie głównych konkluzji, przyczynków oryginalnych oraz propozycje dalszych możliwych kierunków badań.

Rozdział 7 to w istocie cenny załącznik do pracy zawierający m.in. link do otwartego repozytorium na GitHub'ie, bez którego reprodukcja zaprezentowanych wyników badań byłaby niezwykle trudna. Niewątpliwie załącznik ten podnosi wiarygodność zaprezentowanych wyników badań.

W pracy zacytowano 173 publikacje, w tym cztery z udziałem Autora rozprawy [1–4].

Podsumowując, rozprawa mgra Fejzera stanowi ciekawą i cenną dysertację o ważnych, z praktycznego punktu widzenia, celach badawczych i paru oryginalnych przyczynkach w postaci autorskich rozwiązań pojawiających się w rozdziałach 4 i 5. Autor korzysta z osiągnięć, zbiorów danych i pomysłów innych badawczy, umiejętnie znajduje ich słabe strony i skutecznie proponuje własne rozwiązania próbujące zaadresować zidentyfikowane problemy (także w zakresie krytycznej analizy istniejących zbiorów danych). Wykorzystanie projektów o otwartych źródłach, otwartych zbiorów danych, udostępnianie kodu źródłowego własnych rozwiązań, szczegółowe badania empiryczne oraz starania, by reprodukcja badań była możliwa, niewątpliwie stanowią ważny walor recenzowanej dysertacji.

4 Oryginalny wkład naukowy

Za najważniejsze, oryginalne elementy rozprawy mgra Fejzera uważam przede wszystkim:

- Metodę rekomendacji recenzentów stosowaną na etapie przeglądu propozycji zmian kodu, a w szczególności propozycję modelu bazującego na profilach recenzentów, które są

aktualizowane w momencie komentowania zestawu zmian, co pozwala rozwiązać problem przetwarzania całej historii projektu. Konkurencyjne rozwiązania wymagają wczytywania oraz przetwarzania całej historii projektu, co powoduje niepotrzebne zużycie zasobów oraz czasu. W rezultacie zaproponowane rozwiązanie może być zastosowane na dużo większych repozytoriach kodu niż analizowane rozwiązania konkurencyjne. Autor zaproponował cztery warianty swojej metody i porównał je (na zbiorze danych Thongtanunam et al. [5]) z zaimplementowaną przez siebie wersją metody Revfinder (gdyż oryginalna implementacja nie była dostępna). Zaproponowana przez Autora metoda wykorzystująca indeks Tverskiego jako funkcję podobieństwa pomiędzy profilem, a recenzowaną zmianą, okazała się bardzo dobrym rozwiązaniem i osiągnęła najlepsze rezultaty wg miar precision-to-recall oraz F1-measure, a ponadto zużywała mniej zasobów obliczeniowych niż Revfinder.

- Metodę adaptacyjnego podejścia do lokalizacji błędów, które nie wymaga manualnego dostrajania algorytmu, bo algorytm dobiera parametry automatycznie, w ramach uczenia. Podejście to, na tle konkurencyjnego rozwiązania Ye i in. [6] (opublikowanego w 2016 roku w najbardziej prestiżowym czasopiśmie inżynierii oprogramowania *IEEE Transactions on Software Engineering*), w wielu przypadkach sprawuje się bardzo dobrze (np. w przypadku miar $Accuracy@1$, MAP oraz MRR, na zbiorze Ye i in. uzyskuje lepsze rezultaty na wszystkich badanych projektach, por. Rys. 5.3 i Tab. 5.6 dysertacji). Warto docenić także, że Autor przygotował również rozszerzony zbiór danych, który zawiera obliczenia pośrednie oraz gotowe cechy, poprawił brakujące opisy błędów oraz cechę podobieństwa bazującego na API oraz udostępnił kompletny kod źródłowy proponowanej metody.

Ogólnie można stwierdzić dużą wartość praktyczną (techniczną) rozprawy. Warto też zwrócić uwagę, że Autor zawarł w pracy również pewną otoczkę w postaci notacji, które formalizują analizowane problemy (co jest mile widziane w pracach z dyscypliny informatyka) oraz przeprowadził dość szeroko zakrojone badania empiryczne starając się zadbać o możliwość reprodukcji zaprezentowanych badań, co uważam za niezwykle cenne.

5 Uwagi polemiczne

Przedłożona rozprawa zawiera oryginalne propozycje rozwiązań ważnych problemów naukowych i stanowi cenny wkład do wykorzystania metod eksploracji repozytoriów kodu na potrzeby lokalizacji błędów oraz rekomendacji recenzentów kodu. Niemniej jednak w trakcie lektury pracy nasunęły się pewne uwagi i pytania:

1. Ponieważ oceniana praca poświęcona jest głównie tym dwóm kluczowym zagadnieniom, nie sposób oprzeć się wrażeniu, że związek tych dwóch elementów nie został w pracy dobrze uzasadniony. Podjęta przez Autora próba powiązania ich poprzez jakość oprogramowania, nie jest do końca przekonująca, gdyż w dysertacji na próżno szukać pogłębionej dyskusji na temat jakości oprogramowania, stosowanych modeli jakości, czy szeregu norm ISO związanych np. z zewnętrzną i wewnętrzną jakością produktu, czy jakością procesu wytwórczego. Oczywiście powszechnie znane są stwierdzenia, które Autor mógłby spróbować przywołać w swojej obronie np. (przypisywane prof. Barbarze Kitchenham), że jakość oprogramowania jest trudna do zdefiniowania, niemożliwa do zmierzenia, ale łatwa do rozpoznania (“*hard to define, impossible to measure and easy to recognize*”) lub że jakość, podobnie jak piękno, jest

pojęciem względnym, każdy widzi je inaczej ("*quality, like beauty, is very much in the eyes of the beholder*"), ale to nie wystarczy, jeśli jakość oprogramowania ma być spoiną dysertacji. Na podstawie lektury pracy, można jednak obronić tezę, że to co spina dwa kluczowe problemy badawcze podjęte w dysertacji, to: tematycznie – inżynieria oprogramowania, a w obszarze wykorzystywanych metod – techniki wyszukiwania informacji i uczenia maszynowego używane do eksploracji repozytoriów kodu.

2. Niemal zupełnie został pominięty temat zagrożeń dla wiarygodności badań (ang. *threats to validity*). Pojawia się on jedynie w kontekście ostatniego rozważanego problemu badawczego (tj. lokalizacji błędów) w rozdziale 5. Jak już wspomniałem wcześniej, jeśli kolejność rozdziałów wynika z chronologii badań, to rozdział 5 pokazuje postęp jaki poczynił Doktorant w trakcie rozwiązywania kolejnych problemów badawczych. Dostrzegł On bowiem potrzebę dyskusji zagrożeń dla wiarygodności badań. Szkoda jednak, że sama dysertacja w poprzednich rozdziałach nie została uzupełniona o dyskusję zagrożeń dla wiarygodności badań, próbę oceny ich wpływu na wyniki badań oraz próbę minimalizacji/zaadresowania tych zagrożeń. Nawet w samym obszarze inżynierii oprogramowania jest sporo pozycji omawiających te kwestie (np. [7, 8]). Z oceną wpływu istniejących zagrożeń dla wiarygodności badań oraz próbą ich minimalizacji związany jest też drobny problem również w rozdziale 5. Np. pojawia się zagrożenie wynikające z homogeniczności analizowanych projektów (ang. *Projects homogeneity*). Autor świetnie identyfikuje to zagrożenie, ale już nie podejmuje próby jego minimalizacji, ograniczając się tylko do stwierdzenia "*Therefore, this is a valid argument that a more diverse dataset is needed...*".

3. Przeglądy literatury są przeprowadzane metodą określaną niekiedy mianem *ad hoc*, co daje mniejszą pewność, że Autor w pełni zidentyfikował aktualny stan sztuki w każdym z rozpatrywanych problemów badawczych, niż gdyby zostały wykorzystane metody rekomendowane w ramach dynamicznie rozwijającego się (od 2004 roku) nurtu *Evidence-Based Software Engineering* [9, 10] np. metodę systematycznego przeglądu literatury (ang. *Systematic Review/Systematic Literature Review*). Systematyczny przegląd jest formą badania drugiego rzędu (ang. *secondary study*), czyli badania, które poddaje przeglądowi wszystkie podstawowe badania/artykuły naukowe (ang. *primary studies*) odnoszące się do konkretnych pytań badawczych w celu uzyskania pełnego obrazu stanu badań. Systematyczny przegląd wyróżnia to, że używa dobrze zdefiniowanej metodyki do identyfikacji, analizy i interpretacji wszystkich dostępnych źródeł informacji odnoszących się do konkretnych pytań badawczych w maksymalnie bezstronny i powtarzalny (reprodukowalny) sposób. Systematyczne przeglądy (mające swoje źródła w medycynie i nurcie nazwanym *Evidence-Based Medicine*) przeniesione zostały na grunt inżynierii oprogramowania przez prof. Barbarę Kitchenham. Szybko stały się one fundamentem jednego z najważniejszych nurtów inżynierii oprogramowania ostatnich lat, czyli wspomnianego wcześniej *Evidence-Based Software Engineering*. Systematyczne przeglądy są publikowane przez najbardziej prestiżowe czasopisma inżynierii oprogramowania, takie jak *IEEE Transactions on Software Engineering* czy *Information and Software Technology*. Można przypuszczać, że zastosowanie systematycznego przeglądu do opisania istniejących metod i rozwiązań pomogłoby uzyskać maksymalnie kompletny opis stanu sztuki w badanych obszarach dzięki wykorzystaniu dopracowanej metody przeprowadzania przeglądu. Stanowiłoby to wartość samą w sobie, choć należy zaznaczyć, że wymaga to zaangażowania więcej niż samego Autora, co jednak w przypadku wszystkich publikacji Doktoranta, które są wieloautorskie, nie wydaje się kluczowym problemem. Niestety też tego typu przeglądy są niezwykle pracochłonne.

4. Autor pisze o możliwości zastosowania metody rekomendowania recenzentów na dużych repozytoriach np. na GitHubie. Wydawałoby się zatem naturalnym porównanie proponowanego rozwiązania nie tylko z narzędziami opisywanymi w niektórych publikacjach, ale również dostępnymi w największych otwartych repozytoriach kodu na świecie, takich jak wspomniany GitHub, GitLab, czy Bitbucket i wykorzystywanymi przez miliony inżynierów oprogramowania. Dość powiedzieć, że sam GitHub ma ponad 40 milionów użytkowników i ponad 100 000 milionów repozytoriów. Prawdopodobnie zatem nie powinien nas zdziwić fakt, że GitHub wykorzystuje już algorytm rekomendowania recenzentów kodu. Po utworzeniu tzw. *pull request'a* jest możliwość poproszenia o recenzje propozycji zmiany kodu, a GitHub podpowiada kogo warto zaprosić. Podobna sytuacja ma miejsce w przypadku innych wielkich repozytoriów kodu. Bitbucket ma narzędzie, które również sugeruje recenzentów wykorzystując dwa algorytmy bazowe oraz udostępniając możliwość dodawania własnych algorytmów. Również GitLab oferuje podobne rozwiązanie. Cennym byłoby zatem porównać swoje rozwiązanie z tymi powszechnie wykorzystywanymi narzędziami, być może na nowszych, bardziej heterogenicznych projektach/zbiorach danych (a przynajmniej o tych narzędziach wspomnieć). Porównanie takie wydaje się ciekawym kierunkiem dalszych badań.

5. Doktorant przyznaje wielokrotnie, że w pracy występuje problem niezbalansowania klas. Literatura dotycząca metod radzenia sobie z problemem niezbalansowania klas jest niezwykle bogata. Wydaje się, że warto było sięgnąć do niej głębiej, by poszukać bardziej wyrafinowanych metod. Również niektóre miary oceny modeli predykcji nie są zalecane przy znacznym niezbalansowaniu klas (np. *accuracy*). Ten temat również wydaje się interesującym kierunkiem dalszych badań.

6. Drobiazgi:

   (a) Można odnieść wrażenie, że przyjęte notacje mogą niekiedy upraszczać analizowany problem. Przykładowo założenie, że dla każdego błędu istnieje pojedynczy stan przed aplikacją poprawki (ang. *fix*) i po niej (na str. 49). Aplikacja poprawki nie zawsze jest pojedynczą zmianą, zmiany z nią związane mogą następować w odstępach czasu, a pomiędzy nimi mogą zachodzić inne zmiany np. związane z innymi poprawkami.

   (b) Niektóre wartości parametrów (np. pół roku, 2500 commitów) są przyjęte dość arbitralnie, bez głębszego uzasadnienia.

   (c) Zbiory danych nie są niestety szczegółowo opisane. Można oczywiście uznać, że są to publicznie dostępne zbiory danych i można do nich zajrzeć i spróbować samodzielnie przeanalizować, ale to jednak trochę utrudnia odbiór pracy.

   (d) Na str. 8, Autor wymienia cztery rodzaje uczenia maszynowego (*unsupervised learning*, *reinforcement learning*, *supervised learning* i *semi-supervised learning*), a w kolejnym zdaniu pisze, że w dysertacji skupi się na dwóch (*supervised learning* i *learning to rank*), z których tego ostatniego nie ma na poprzedniej liście.

   (e) Np. w rozdziale 4.5.2 jest mowa o tzw. "*accurate reviewer recommendations*". Warto dopisać kiedy przydzielenie recenzenta jest uznane za "*accurate*", a nawet wcześniej szczegółowo omówić kiedy rekomendacje są uznawane za poprawne.

   (f) Skróty (np. VST) warto powiązać z ich rozwinięciem przy pierwszym użyciu.

   (g) W rozdziale 3.3 mamy "*We used the GHTorrent dataset [41], which contains data of 90 GitHub projects and their forks, out of which we choose 42 projects containing enough issues*

and commit comments for our purposes." – nie całkiem jest jasne ile to jest "*enough*" i dlaczego właśnie tyle.

(h) Czy metoda zaprezentowana w rozdziale 5 (ang. *adaptive bug localization*) będzie równie dobrze działała, gdy w raportach błędów zabraknie śladu stosu (ang. *stack trace*)? Czy cechy $\phi_7 ... \phi_{14}$ nie wymagają klas, zmiennych, pakietów w opisie zgłoszenia?

Sformułowane uwagi polemiczne i pytania nie podważają ogólnie pozytywnego odbioru pracy.

6 Konkluzje

Stwierdzam, że rozprawa doktorska Pana mgra Mikołaja Fejzera, stanowiąc oryginalne rozwiązanie podjętych problemów naukowych oraz wykazując głęboką wiedzę Doktoranta w dyscyplinie informatyka, a także umiejętność samodzielnego prowadzenia zaawansowanych badań naukowych, spełnia wymagania określone w stosownej Ustawie. Zakres i jakość badań naukowych przeprowadzonych przez Autora potwierdzają jego znaczący i oryginalny wkład. Dodatkowo Doktorant opublikował osiągnięte wyniki w trzech publikacjach, m.in. w artykule, który ukazał się w czasopiśmie *Journal of Intelligent Information Systems* z listy JCR, w materiałach konferencji BDAS oraz ADBIS (z listy CORE o rankingu B), a ponadto jedna praca jest w trakcie recenzji w czasopiśmie. W mojej ocenie dorobek publikacyjny Kandydata można uznać za wystarczający. Wobec powyższego niniejszym wnioskuję o dopuszczenie rozprawy do publicznej obrony.

Bibliografia

[1] Mikołaj Fejzer, Michał Wojtyna, Marta Burzańska, Piotr Wiśniewski i Krzysztof Stencel. "Open Source Is a Continual Bugfixing by a Few". W: *Advances in Databases and Information Systems*. Red. Yannis Manolopoulos, Goce Trajcevski i Margita Kon-Popovska. Cham: Springer International Publishing, 2014, s. 153–162. ISBN: 978-3-319-10933-6.

[2] Mikołaj Fejzer, Michał Wojtyna, Marta Burzańska, Piotr Wiśniewski i Krzysztof Stencel. "Supporting Code Review by Automatic Detection of Potentially Buggy Changes". W: *Beyond Databases, Architectures and Structures*. Red. Stanisław Kozielski, Dariusz Mrozek, Paweł Kasprowski, Bożena Małysiak-Mrozek i Daniel Kostrzewa. Cham: Springer International Publishing, 2015, s. 473–482. ISBN: 978-3-319-18422-7.

[3] Mikołaj Fejzer, Piotr Przymus i Krzysztof Stencel. "Profile based recommendation of code reviewers". W: *Journal of Intelligent Information Systems* 50 (2018), s. 597–619.

[4] Mikołaj Fejzer, Jakub Narebski, Piotr Przymus i Krzysztof Stencel. "Tracking Buggy Files: New Efficient Adaptive Bug Localization Method". W: *manuscript under review* 50 (2020), s. 597–619.

[5] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida i K. Matsumoto. "Who should review my code? A file location-based code-reviewer recommendation approach for Modern Code Review". W: *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 2015, s. 141–150.

[6] X. Ye, R. Bunescu i C. Liu. "Mapping Bug Reports to Relevant Files: A Ranking Model, a Fine-Grained Benchmark, and Feature Evaluation". W: *IEEE Transactions on Software Engineering* 42.4 (2016), s. 379–402.

[7] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell i Anders Wesslén. *Experimentation in Software Engineering: An Introduction*. 2nd. Berlin Heidelberg, Germany: Springer, 2012. ISBN: 978-3-642-29043-5.

[8] Lech Madeyski. *Test-Driven Development: An Empirical Evaluation of Agile Practice*. Foreword by Prof. Claes Wohlin. (Heidelberg, London, New York): Springer, 2010. ISBN: 978-3-642-04287-4. DOI: 10.1007/978-3-642-04288-1 URL: http://dx.doi.org/10.1007/978-3-642-04288-1.

[9] Barbara A. Kitchenham, Tore Dybå i Magne Jørgensen. "Evidence-Based Software Engineering". W: *ICSE'04: International Conference on Software Engineering*. 2004, s. 273–281.

[10] Barbara Kitchenham, David Budgen i Pearl Brereton. *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press, 2016.

.....
Lech Madeyski