Politechnika Wrocławska

Wydział Informatyki i Zarządzania
Katedra Inżynierii Oprogramowania

Prof. dr hab. inż. Zbigniew Huzar

Wrocław, 12 sierpnia, 2019

Recenzja rozprawy doktorskiej mgr inż. Wojciecha Frącza

Rozprawa doktorska: Wojciech Frącz, *Modelowanie jakości kodu źródłowego na podstawie danych gromadzonych w systemach kontroli wersji*. Maszynopis. Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie. Wydział Informatyki, Elektroniki i Telekomunikacji, Katedra Informatyki. Kraków 2019 (161 stron, 105 pozycji bibliograficznych).

Recenzja sporządzona na podstawie pisma Prodziekana Wydziału, prof. n. dr hab. inż. Krzysztofa Winczy, z dnia 17 czerwca 2019.

Zakres i cel rozprawy

Przedmiotem rozważań rozprawy jest czytelność kodu rozumiana jako jedna z zasadniczych cech jego jakości. Stawia się następującą tezę rozprawy:

> *Opracowanie jakościowego modelu kodu źródłowego z użyciem technik uczenia maszynowego na podstawie danych gromadzonych w systemach kontroli wersji pozwoli na wykazanie większej trafności w rozpoznawaniu kodu niskiej jakości niż dostępne obecnie narzędzia wykorzystujące jego statystyczną analizę.*

Ocena jakości kodu jest w rozprawie utożsamiana z jego czytelnością. Takie utożsamienie jest zawężeniem pojęcia jakości oprogramowania – liczne normy jakości oprogramowania, np. rekomendowane przez historyczną już normę ISO 9126 oraz zastępująca ją normę ISO/IEC 25010:2011, wskazują, że czytelność jest tylko jedną z wielu charakterystyk jakości, najczęściej jest wymieniana jako jedna z podcharakterystyk utrzymywalności (*maintainability*). Być może lepiej byłoby to zaznaczyć w tytule rozprawy mówiąc o modelowaniu czytelności kodu.

Potrzeba oceny czytelności kodu ma pełne uzasadnienie w procesach wytwarzania oprogramowania; powtórne wykorzystanie kodu przy realizacji nowych projektów lub konieczność modyfikacji kodu realizowanego projektu wynikająca ze zmiany wymagań są typowymi przykładami takich sytuacji.

Z tezy rozprawy wynika, że jej celem jest opracowanie metody oceny czytelności kodu źródłowego. Rozprawa tego celu nie określa jawnie, gdyż nie są jej celami te, które wskazuje tekst rozprawy. To, co rozprawa określa jako *cele* (strona 5) są to bowiem *zadania*, które należy zrealizować, aby osiągnąć cel.

Zakres rozprawy, przyjęta teza i jej cel, niezależnie od wyżej sformułowanych zastrzeżeń, są dobrze uzasadnione praktycznymi potrzebami w procesach rozwoju oprogramowania.

Zawartość rozprawy, uwagi ogólne

Rozprawa składa się z ośmiu rozdziałów i sześciu dodatków.

Rozdział 1. wyjaśnia tezę i cele rozprawy. Nie wprowadza natomiast pojęcia problemu badawczego, od którego zwykle powinna brać początek każda rozprawa.

Rozdział 2. analizuje pojęcie czytelności w kontekście refaktoryzacji kodu. Biorąc pod uwagę ten kontekst można się zastanawiać czy chodzi tylko o czytelność czy też o analizowalność, bądź ogólniej o utrzymywalności kodu. Pojęcie czytelności nie jest w pracy sprecyzowane, nie ma też wprowadzonych miar czytelności. Punkt 2.1 jakość kodu źródłowego określa w sposób enigmatyczny *jako jego cechę, mówiącą o tym jak dobrze jest on napisany*. Zwracając uwagę na brak możliwości obiektywnego zdefiniowania takiej cechy, kojarzy się czytelność ze stosowanymi obiektywnymi metrykami oceny kodu i, ostatecznie, pomimo tych trudności, wskazuje się na potrzebę i możliwość automatycznego przeglądu czytelności kodu w oparciu o uczenie maszynowe. Stwierdzenie, że w literaturze jest brak propozycji jakościowego modelu kodu źródłowego jest ważne, ale nie jest poparte systematycznym przeglądem literatury, na przykład zgodnym z szeroko akceptowanym zestawem zaleceń opracowanych przez B. Kitchenham, S. Charters, *Guidelines for performing Systematic Literature Reviews in Software Engineering* (https://userpages.uni-koblenz.de/~laemmel/esecourse/slides/slr.pdf).

Rozdział 3. przedstawia koncepcję jakościowego modelu kodu źródłowego SCQM (*Software Code Quality Model)* oraz metodykę jego wyznaczania. Jakość kodu źródłowego odnosi się do programów pisanych w Javie, a dokładniej jest zawężona do oceny jakości metod o ograniczonej długości. Pojęcie modelu jakościowego jest rozumiane w specyficzny sposób. Model ma tu naśladować programistę, który w oparciu o własne subiektywne odczucia ocenia kod jako dobry lub zły, czyli możliwy do zaakceptowania lub odrzucenia w danym projekcie. Model SCQM klasyfikuje dany program z prawdopodobieństwem p jako dobry i prawdopodobieństwem $(1 - p)$ jako zły. Rozpatruje się dwie wersje modelu: model bezwzględny (a – *absolute*), aSCQM, ocenia czy dany program jest akceptowanej jakości, a model względny (r – *relative*), rSCQM, ocenia czy refaktoryzacja danego programu podnosi jego jakość. Oba modele są implementowane w postaci odpowiednio wytrenowanej sieci neuronowej.

Należy zastrzec, że użyty tu termin 'model jakościowy' odbiega od znaczenia tego terminu nadawanego przez standardy ISO oraz IEEE. Standardy te określają model jakości znacznie szerzej przez wprowadzenie perspektyw (punktów widzenia) ocen, związanych z nimi hierarchiami charakterystyk jakości, miarami przypisywanymi charakterystykom i, wreszcie, kryteriami oceny jakości, które klasyfikują programy czteropoziomowo jako złe (*non-accepted*), dostateczne (*minimal*), dobre (*target*) i znakomite (*exceeded*).

Rozdział 4. dotyczy ważnego zagadnienia ustalenia zbioru kodów źródłowych na których można przeprowadzić eksperymenty prowadzące do nauczenia i zweryfikowania sieci neuronowej. Wybrano funkcjonującą już ponad 10 lat platformę GitHub stanowiąca repozytorium projektów otwartych. Repozytorium przechowuje projekty wraz z historią tworzenia ich kolejnych wersji i, co istotne z punktu widzenia rozprawy, zawiera komentarze dotyczące powodów zmian wersji, a wśród nich refaktoryzację. Dzięki temu, z wybranych, najbardziej popularnych 350 otwartych projektów wyodrębniono prawie 18 tysięcy zmian refaktoryzacyjnych dla ponad 12 tysięcy metod w 16 tysiącach klas.

Po to, aby refaktoryzowane metody (ich treści a nie ciała!) mogły stanowić dane dla sieci neuronowej klasyfikującej ich jakość, była konieczna odpowiednia ich reprezentacja. Przyjęto, że będą one przedstawiane w postaci drzew rozbioru składniowego. Wykorzystując bibliotekę *JavaParser* dla każdej metody wyznaczono jej drzewo rozbioru składniowego, a następnie przedstawiono je w postaci liniowej – ciągu tokenów (symboli) reprezentujących elementy gramatyki formalnej języka Java. Dokonanym wyborom oraz sposobowi reprezentacji danych towarzyszą odwołujące się do pragmatyki przekonujące uzasadnienia.

Rozdział 5. opisuje eksperyment, którego celem była ocena jakościowa kodów i ich zmian refaktoryzacyjnych w wybranym zbiorze kodów. W eksperymencie trwającym sześć miesięcy wzięło udział ponad 500 zdalnych respondentów, którzy sklasyfikowali ponad 5000 przypadków. Efektem było utworzenie tzw. *benchmarku*, czyli zbioru zgromadzonych kodów metod wraz z ich ocenami bezwzględnymi i względnymi.

Benchmark, dostępny na utworzonym przez Autora portalu, zawiera prawie 1300 przypadków metod o określonej jakości. Stanowił on zbiór danych treningowych do uczenia rekurencyjnej sieci neuronowej. Implementacja i uczenie sieci neuronowych są bardzo zwarcie opisane w rozdziale 6. Sieci neuronowe dla obu modeli, aSCQM oraz rSCQM, zostały zaimplementowane w środowisku *TerSCnsorFlow* rozwijanym współcześnie przez grupę badawczą *Google Brain*. Podkreślenia wymaga konieczność zaangażowania sporych zasobów obliczeniowych – 8-12 godzin pracy superkomputera w Cyfronecie. Ostatecznym wynikiem, przy doborze różnych parametrów modelu, było uzyskanie trafności rozpoznania dobrego kodu na poziomie 65% dla modelu aSCQM i 80% dla rSCQM.

Nie są to poziomy nadmiernie wysokie, ale jednak na tyle istotne, że Autor zdecydował się porównać skuteczność opracowanych modeli z innymi dostępnymi metodami oceny jakości kodu. Przeprowadzone eksperymenty, opisane w rozdziale 7., wskazują na wyższość opracowanych modeli aSCQM i rSCQM – modele te klasyfikują z większą kody o dobrej i złej jakości. Rozdział kończy się rozważaniem potencjalnej integracji opracowanego modelu z innymi rozwiązaniami.

Rozprawę kończy rozdział 8. podsumowujący jej osiągnięcia i sugerowane kierunki jej kontynuacji.

Ocena rozprawy

Należy zgodzić się ze szczegółowymi osiągnięciami rozprawy, które, uzasadniając potwierdzenie tezy, Autor wymienia w ostatnim rozdziale. W opinii recenzenta jednak najbardziej wartościowym wynikiem rozprawy jest oryginalny pomysł oceny jakości kodu polegający na odwołaniu się do zgromadzonej wiedzy eksperckiej. Jest to podejście odmienne od ocen opartych na wyliczaniu wartości obiektywnych, jawnie zdefiniowanych metryk, i potencjalnie może być rozważane do wykorzystania w innych fazach wytwarzania oprogramowania.

W proponowanym podejściu do oceny czytelności kodu nie ma jednej, jawnie zdefiniowanej metryki, a zamiast niej jest odwołanie się do opinii ekspertów, których doświadczenie w sposób niejawny wyraża ocenę składni i semantyki kodu. Ostatecznie opinie ekspertów są tu odpowiednikiem tradycyjnie rozumianej miary. Wadą przedstawianego konkretnego podejścia jest zawężenie jakości kodu do czytelności i brak odniesienia do standardów modeli jakości. Zaletą jest natomiast możliwość jego zastosowania w praktyce wytwarzania oprogra-

mowania, chociaż wymaga to jeszcze dodatkowej pracy i, przede wszystkim, znalezienia zainteresowanej firmy programistycznej.

Należy podkreślić, że w realizacji postawionego celu Doktorant sprawnie posługiwał się nowoczesnymi, zaawansowanymi narzędziami obliczeniowymi.

Rozprawa jest napisana przejrzyście zwięzłym i niesformalizowanym językiem. Krótki wstęp na początku i podsumowanie na końcu każdego rozdziału ułatwiają jej zrozumienie.

Niezależnie od wcześniej wskazanych uwag krytycznych rozprawę oceniam bardzo pozytywnie.

Wniosek końcowy

Rozprawa mgr. inż. Wojciecha Frącza przedstawia oryginalną propozycję rozwiązania praktycznego, ważnego problemu w procesie wytwarzania oprogramowania.

Uważam, że rozprawa całkowicie spełnia wymagania stawiane rozprawom doktorskim i tym samym rekomenduję dopuszczenie jej do publicznej obrony.