Dr hab. inż. Jerzy Nawrocki, prof. nadzw.
Instytut Informatyki
Politechnika Poznańska

Poznań, 2019-08-19

Recenzja rozprawy doktorskiej

mgra inż. Wojciecha Frącza

pt.: Modelowanie jakości kodu źródłowego na podstawie danych gromadzonych w systemach kontroli wersji

1. Informacje ogólne

Przedstawiona do recenzji rozprawa składa się z ośmiu rozdziałów. Pierwsze dwa mają charakter wprowadzający (łącznie 25 stron), w rozdz. 3. przedstawiono koncepcję tzw. jakościowego modelu kodu źródłowego, a w kolejnych czterech rozdziałach opisano aspekty związane z konstrukcją i ewaluacją, czyli przygotowanie danych z repozytorium GitHub (rozdz. 4), badanie opinii programistów (rozdz. 5), budowę sieci neuronowej i jej uczenie (rozdz. 6) oraz eksperyment ewaluacyjny (rodz.7). W rozdz. 8 zawarto uwagi końcowe, w tym podsumowanie rezultatów przeprowadzonych badań. Część główna pracy jest uzupełniona ośmioma dodatkami (ostatni z nich zawiera zawodowe CV Autora). Cała rozprawa liczy 161 stron (bez spisu treści i streszczeń).

2. Cel i zakres rozprawy

Ogólnie rzecz biorąc, praca dotyczy zastosowania metod uczenia maszynowego do inspekcji oprogramowania. Inspekcje kodu są – obok testowania – jedną z najważniejszych, z praktycznego punktu widzenia, metod kontroli jakości kodu. Niestety, z różnych względów, o których pisze Autor, programiści niechętnie analizują cudzy kod (między innymi dla tego, że jest to zadanie dość pracochłonne). Metody uczenia maszynowego mogą stanowić ważną pomoc w tym zakresie i z tego punktu widzenia tak zarysowana **tematyka** rozprawy jest nie tylko **bardzo atrakcyjna intelektualnie** (ostatnio metody uczenia maszynowego cieszą się wielką popularnością), ale również **ma znaczenie praktyczne**.

Celem rozprawy – zgodnie ze sformułowaniem przedstawionym przez Autora – było wykazanie następującej tezy (str. 4 rozprawy):

> *Opracowanie jakościowego modelu kodu źródłowego[1] 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 statyczną analizę[2].*

[1] Zamiast terminu „*jakościowy **model kodu źródłowego***" wolałbym zwrot „***model jakości** kodu źródłowego*". Sam Autor używa akronimu SCQM od ang. *Source Code Quality Model*, czyli – zgodnie z regułami języka angielskiego – chodzi o model jakości. Wskazuje na to również zawartość kolejnych rozdziałów rozprawy, w których m.in. modelowaniu podlegają estetyczne preferencje programistów (rozdz. 5, 6.5 i 6.6).

[2] Przeciwstawianie *uczenia maszynowego* metodom analizy statycznej kodu nie za bardzo jest zgodne z ogólnie przyjętym rozumieniem tego terminu (analiza statyczna to analiza bez wykonywania programu, czyli też u.m.).

Mam wrażenie, że takie sformułowanie celu jest trochę zbyt ogólne względem argumentacji przedstawionej w rozprawie. Uwzględniając zawartość rozprawy i przyjęte ograniczenia, cel rozprawy można by przeformułować do następującej postaci:

> Celem badań przedstawionych w rozprawie było zbadanie skuteczności automatycznej oceny jakości kodu źródłowego (rozumianego jako pojedyncza metoda klasy zapisanej w języku Java) za pomocą rekurencyjnych sieci neuronowych wykorzystujących mechanizm LSTM[3], do uczenia których posłużyły dane pozyskane z repozytorium kodu GitHub i opinie programistów.

Autor rozważa dwa konteksty użycia takiej oceny:

- ocenę **kodu źródłowego** (przedmiotem oceny jest pojedyncza wersja metody; jest to tzw. model bezwzględny – aSCQM);
- ocenę **poprawek** wprowadzonych do kodu źródłowego (przedmiotem oceny są dwie wersje tej samej metody: wersja sprzed refaktoryzacji i wersja po refaktoryzacji; jest to tzw. model względny – rSCQM).

Skuteczność rekurencyjnych sieci neuronowych oceniono poprzez porównanie ich trafności oceny z trafnością oceny popularnego narzędzia CheckStyle, Do oceny skuteczności wykorzystano 96 metod zapisanych w języku Java, znajdujących się w opracowanym przez Autora benchmarku.

3. Wkład Autora

Za najbardziej wartościowe elementy rozprawy uważam:

- **Opracowanie narzędzia automatycznie oceniającego jakość kodu źródłowego z wykorzystaniem metod uczenia maszynowego**. Jest to główne osiągnięcie rozprawy. Z pewnością wymagało ono od Autora dużo pracy. Pokazuje też umiejętności programistyczne Doktoranta i znajomość nowoczesnych narzędzi, do których należy zaliczyć m.in. TensorFlow i API związane z repozytorium GitHub.
- **Opracowanie benchmarku zorientowanego na ewaluację narzędzi służących do automatycznej oceny jakości metod zapisanych w języku Java.** Jest to niejako efekt uboczny prac dotyczących narzędzi aSCQM i rSCQM. Zgadzam się z Autorem, że taki benchmark może się przydać również w nieco innym kontekście i dobrze, że Autor go udostępnił.
- **Przeprowadzenie eksperymentu ewaluacyjnego.** W eksperymencie wzięło udział ponad 500 osób. Eksperyment dostarczył wiedzy na temat opracowanego przez Autora narzędzia, ale też jest źródłem wiedzy dotyczącej m.in. „dostrajania" narzędzia TensorFlow do potrzeb związanych z analizą kodu źródłowego i sposobu przygotowania kodu źródłowego do analizy za pomocą rekurencyjnych sieci neuronowych.

Niestety, nie wszystko było dla mnie jasne i przekonujące. Oto najważniejsze wątpliwości, jakie pojawiły się w trakcie czytania rozprawy:

- Formułując cel rozprawy Autor użył sformułowania „*jakościowy model kodu źródłowego*" ale **nie określił co rozumie przez jakość kodu źródłowego**. Czytając początkowe rozdziały można odnieść wrażenie, że chodzi o brak przykrych zapachów w kodzie lub o łatwość zrozumienia kodu. W końcu okazuje się, że nie chodzi ani o jedno ani o drugie. Jeśli się nie mylę, to jakość jest tutaj mieszanką wrażeń estetycznych programisty (rozdz. 5) i wiary w postępowość zmian (jeśli w *commit message* wystąpiło słowo *refactor* lub *readability* i w kodzie metody dokonano zmiany, to jej nowa wersja jest uznawana za lepszą – rozdz. 4). Wygląda to zbyt skomplikowanie.

[3] ang. *Long Short-Term Memory* – Mechanizm umożliwiający uczenie się „długoterminowych" zależności między danymi.

- Mam wątpliwości, co do **użyteczności informacji** dostarczanej przez opracowane narzędzie. Przedmiotem oceny są pojedyncze metody zapisane w języku Java. Opracowane narzędzie pozwala automatycznie wyselekcjonować metody „niskiej jakości". Programista dowie się, że te metody nie spodobały się systemowi oceniającemu, ale nie dowie się dlaczego. Gdyby opracowane narzędzie identyfikowało przykre zapachy w kodzie lub naruszenia standardu kodowania, to takie objaśnienie byłoby możliwe, ale w przypadku omawianego narzędzia jest to niemożliwe. System oceniający krytykuje metody, ale jest to krytyka niekonstruktywna.
- Mam też **wątpliwość natury metodologicznej**. Próbka ucząca zawierała przykładowe metody wraz z ich „etykietami" określającymi jakość danej metody. W przypadku etykiet pozyskanych od programistów prezentowano im dwie wersje danej metody i mieli określić, która z nich jest „lepsza" (*Which code is better*?). Jeśli trzech programistów oceniło, że dana wersja metody jest lepsza od tej drugiej, to ta wersja dostawała etykietę „dobra jakość". Problem polega na tym, że **jedna wersja może być lepsza od drugiej, co nie oznacza, że ta pierwsza jest dobra** (obie mogą być niskiej jakości) ani też, że ta druga jest niskiej jakości (obie mogą być dobrej jakości). Przyjęta metoda etykietowania wydaje się tego zjawiska nie uwzględniać.

  Inny problem o charakterze metodologicznym związany jest z **pomijaniem nazw zmiennych w próbkach uczących**. Skoro programistom prezentowano metody w wersji oryginalnej wraz z nazwami zmiennych, to mogło się zdarzyć, że decyzja o tym, która wersja bardziej się podoba wynikała m.in. z użytych nazw. Po usunięciu nazw tej różnicy już nie było (był tylko token NAZWA bez konkretnej nazwy) i sieć neuronowa nie mogła tej różnicy w użytych nazwach wziąć pod uwagę.

- W rozprawie **zabrakło analizy statystycznej**. Jak rozumiem, próbka testująca była cały czas ta sama, a można było ją losować, co pozwoliłoby uniezależnić się od konkretnego podziału na próbkę uczącą i testującą. Brakuje też analizy wielu innych zagrożeń dla otrzymanych wyników, o których mowa np. w książce C. Wohlina et al. „*Experimentation in Software Engineering*", Springer, 2012 (głównie rozdz. 8.8).
- **Sposób porównania opracowanego narzędzia z narzędziem CheckStyle budzi pewne zdziwienie**, gdyż Autor sprowadził wykorzystanie CheckStyle'a do pomiaru trzech metryk: złożoności cyklomatycznej, NPath i NCSS. Jeśli chodzi o złożoność cyklomatyczną, Autor arbitralnie przyjął, że **każda metoda, która ma złożoność cyklomatyczną 2 lub więcej jest niskiej jakości**, co oznacza, że każda metoda, która nie ma żadnych rozgałęzień jest „dobrej jakości", a wprowadzenie jakiejkolwiek instrukcji warunkowej lub pętli powoduje, że metoda staje się „niskiej jakości". Oczywiste jest, że w ten sposób nie da się odzwierciedlić wrażeń estetycznych programistów i w tym badaniu CheckStyle był od początku skazany na przegraną. Fakt, że trafność klasyfikacji w tym przypadku była na poziomie 56% (czyli powyżej 50%, które należałoby oczekiwać od zwykłego podejścia losowego) należałoby traktować jako przypadek losowy.

4. Wiedza Kandydata

Przedstawiona do oceny rozprawa pokazuje, że Autor posiada głęboką wiedzę z zakresu inżynierii oprogramowania i technik uczenia maszynowego. Kandydat jest bardzo dobrze przygotowany do prowadzenia nowoczesnych badań w zakresie inżynierii oprogramowania (i nie tylko tam).

To czego mi ewentualnie zabrakło w rozprawie, to przegląd literatury (najlepiej w formie *Systematic Literature Review*) dot. zastosowań technik uczenia maszynowego w przeglądach kodu (ang. *code reviews*). Doktorant dysponuje bardzo szeroką wiedzą w tym zakresie, ale jednak pewne (jak mi się wydaje, dość ważne) prace umknęły jego uwadze:

- Ghulam Rasool, Zeeshan Arshad: *A review of code smell mining techniques*, Journal of Software Evolution and Process, 2015 (27), str. 867–895.

  Jest to praca przeglądowa, pokazująca m.in. własności narzędzia CheckStyle na tle paru innych narzędzi (PMD i JDeodorant). Jej znaczenie dla rozprawy wynika z faktu, że Autor ograniczył się tylko do narzędzia CheckStyle, a ta praca sugeruje, że warto byłoby uwzględnić także inne narzędzia (chociażby wspomniane PMD i JDeodorant), gdyż mają one komplementarne cechy względem CheckStyle.

- Francesca Arcelli Fontana, Mika V. Mäntylä, Marco Zanoni, Alessandro Marino: *Comparing and experimenting machine learning techniques for code smell detection*, Empirical Software Engineering, June 2016, Volume 21, Issue 3, pp 1143–1191.

  W tej pracy porównano eksperymentalnie skuteczność wykrywania czterech przykrych zapachów (*Data Class, Large Class, Feature Envy, Long Method*) za pomocą 16 narzędzi uczenia maszynowego. Zwraca uwagę staranność metodologiczna przeprowadzonych badań, która mogłaby być wzorem postępowania dla wielu osób, w tym także dla Autora rozprawy.

Mimo tej uwagi **jestem pod bardzo pozytywnym wrażeniem co do wiedzy Doktoranta**, którą zaprezentował w rozprawie (w tym w rozdz. 2).

5. Inne uwagi

Praca jest napisana w języku polskim. **Jestem pod wrażeniem staranności językowej**. Pracę czytało mi się bardzo przyjemnie. Oczywiście, w tak obszernym opracowaniu zawsze znajdzie się jakiś błąd edytorski. Tak też jest i w tym przypadku. Są drobne literówki, jak np. „wyzwania stawianie rozprawie" (str. 6), czy „poddane ocenie przed ... narzędzia" (str. 86; powinno być „przez" zamiast „przed"). Zdarzyło się też powtórzenie informacji (np. na str. 67 i na str. 70 jest ta sama uwaga nt. komentarza studenta dot. koloru tekstu) i żargon (na str. 63 jest wyraz „wykomentowywanie").

Niestety, zdarzył się też błąd merytoryczny (na szczęście nie dotyczy on badań przeprowadzonych przez Autora). Na stronie 23 Autor napisał:

> *Model rozumiejący język programowania jest jednak w stanie nie tylko wykryć **błąd syntaktyczny**, ale także zasugerować odpowiedni opis błędu programiście [83]. Takie rozwiązania już brzmią zdecydowanie sensowniej i **wykraczają poza możliwości kompilatorów**.*

Błąd polega na tym, że *de facto* automatyczna poprawa błędów składniowych przez kompilatory **jest możliwa**. Oto dwa przykładowe artykuły na ten temat:

- J.P. Levy „*Automatic correction of syntax-errors in programming languages*", Acta Informatica, vol. 4, iss. 3, **1975**, 271-292.
- Tanaka and King-Sun Fu, „*Error-Correcting Parsers for Formal Languages*", IEEE Transactions on Computers, **1978**, Volume C-27, Number 7, 605-616.

6. Podsumowanie

Uważam, że omawiana rozprawa stanowi oryginalne rozwiązanie problemu naukowego, jakim jest zbadanie skuteczności automatycznej oceny jakości kodu za pomocą rekurencyjnych sieci neuronowych, w szczególności jakości rozumianej jako preferencje estetyczne programistów w odniesieniu do kodu źródłowego. Autor wykazał się dużą wiedzą z zakresu inżynierii oprogramowania i sieci neuronowych. Pewne dyskusje i argumenty występujące w rozprawie wymagałyby jeszcze dopracowania (zwłaszcza jeśli miałyby być przedmiotem dalszej publikacji), ale ogólnie rzecz biorąc rozprawę oceniam pozytywnie i **wnoszę o dopuszczenie Doktoranta do dalszych etapów przewodu doktorskiego**.