prof. dr hab. inż. Janusz Górski
Profesor emeritus Politechniki Gdańskiej
ul. Nowogródzka 25
80-124 Gdańsk

Recenzja rozprawy doktorskiej mgr Michała Mnicha:
Testowanie mutacyjne – optymalizacja procesu i praktyczne zastosowania

1. Problem i teza rozprawy

Celem recenzowanej rozprawy było zbadanie możliwości usprawnienia procesu testowania mutacyjnego pod kątem jego skuteczności i wydajności.

Autor dokonał prezentacji aktualnego stanu badań w obszarze problematyki testowania mutacyjnego i na tym tle przedstawił własne tezy, które zamierzał uzasadnić w trakcie badań.

Testowanie mutacyjne ma na celu poprawę jakości testowania programu, który powstaje w ramach procesu rozwojowego oprogramowania. Odnosi się więc do obszaru, który jest (w większym lub mniejszym stopniu) obecny w praktycznie stosowanych procesach wytwarzania i rozwoju oprogramowania, to znaczy do testowania powstającego i pielęgnowanego produktu programistycznego. Istota testowania polega na dokonywaniu eksperymentów z powstającym produktem – programem komputerowym. Eksperymenty te mają na celu sprawdzenie zgodności funkcjonowania programu ze sformułowanymi względem niego oczekiwaniami (wyrażonymi jakąś formą specyfikacji tych oczekiwań). Stwierdzona zgodność potwierdza, że w warunkach określonych danym eksperymentem (przypadkiem testowym) program funkcjonuje poprawnie. Niezgodność oznacza awarię w funkcjonowaniu programu i wskazuje, że w programie występuje defekt (będący przyczyną tej awarii). Stwierdzenie awarii stanowi punkt wyjścia do poszukiwania defektu i jego usunięcia, co może prowadzić do poprawy niezawodności programu (jego zdolności do funkcjonowania zgodnie ze specyfikacją). Tak więc zdolność do ujawniania defektów jest ważną cechą zestawu testów i świadczy o ich właściwym doborze (dobrej jakości testów). Jest to podstawowa intencja, z którą są konstruowane przypadki testowe. Jednak trudno oczekiwać, że tworzone z tą intencją testy będą 'doskonałe' ze względu na ogromną różnorodność w przestrzeni możliwych defektów oprogramowania oraz brak wiedzy na temat mechanizmów ich materializacji (defekty oprogramowania są skutkiem pomyłek człowieka-programisty, a wiedza na temat „błędu ludzkiego" jest wciąż bardzo ograniczona). Jedyną pewną wiedzą w tym zakresie jest to, że w nietrywialnych programach (a więc takich o znaczeniu przemysłowym) defekty oprogramowania z pewnością występują. Pozostaje więc poszukiwanie sposobów tworzenia jak najskuteczniejszych zestawów testowych oraz, dla już opracowanych zestawów, poszukiwanie metod służących ich poprawie.

Testowanie mutacyjne mieści się w obszarze drugiego z powyższych celów. Idea polega na tym, aby w przetestowanym już programie 'zasiać' dodatkowe defekty (nazywane mutantami) i sprawdzić czy istniejący zestaw testów jest zdolny do ich wykrycia. Wykrycie defektu-mutanta oznacza jego 'zabicie' i potwierdza jakość obecnego zestawu testów. Niewykrycie mutanta oznacza, że testy nie są doskonałe i daje podstawę do ich poprawy (uzupełnienie o test wykrywający danego mutanta). W ten sposób mamy do czynienia z procesem doskonalenia wyjściowego zestawu testów.

Tak rozumiany obszar testowania mutacyjnego rodzi wiele pytań, na przykład o zasady doboru mutantów, o miejsce testowania mutacyjnego w cyklu rozwojowym oprogramowania, ale przede wszystkim o skuteczność oraz o wydajność tego podejścia (innymi słowy o stosunek korzyści względem poniesionych nakładów).

Autor w swoich badaniach skoncentrował się na zarządzaniu procesem testowania mutacyjnego, a w przedstawionych tezach zaproponował szereg usprawnień tego procesu. Sformułował 5 tez badawczych (przytoczonych *in extenso* poniżej):

1. Efektywność testowania mutacyjnego nie zmniejsza się znacząco, jeśli w nowej wersji oprogramowania mutanty są generowane wyłącznie dla kodu nowego bądź zmodyfikowanego.
2. Zmiana rozkładu prawdopodobieństwa losowania operatorów mutacyjnych, dokonana przy użyciu podejścia bayesowskiego, może być wykorzystana do istotnego zmniejszenia liczby mutantów przy dopuszczalnej niewielkiej utracie efektywności procesu.
3. Wzbogacenie podejścia Test-Driven Development o testowanie mutacyjne znacząco podnosi charakterystyki jakościowe kodu i zwiększa niezależność testowania poprzez obniżenie zjawiska stronniczości (ang. bias) autora testowanego kodu.
4. Podejście polegające na wprowadzeniu wielu mutacji do jednej kompilacji ma swoje ograniczenia i nie każdy operator mutacyjny może współistnieć z wszystkimi innymi.
5. Podejście polegające na wprowadzeniu wielu mutacji do jednej kompilacji dla języków, w których modyfikacja kodu pośredniego nie jest możliwa może znacząco przyspieszyć proces testowania mutacyjnego.

Takie sformułowanie tez badawczych nasuwa następujące refleksje.

1. Każdy z zaadresowanych w pracy problemów jest stosunkowo 'autonomiczny' względem pozostałych (z wyjątkiem problemów objętych tezami 4 i 5, które dotyczą mutantów objętych wspólną kompilacją).
2. Tezy 4 i 5 można było wyrazić jedną tezą, gdyż dotyczą wspólnego problemu: mutacji objętych wspólną kompilacją. Teza ta mogłaby być na przykład sformułowana następująco: *Dla języków gdzie modyfikacja kodu pośredniego nie jest możliwa, podejście polegające na wprowadzeniu wielu mutacji do jednej kompilacji może znacząco przyspieszyć proces testowania mutacyjnego, przy czym dobór operatorów mutacyjnych uczestniczących we wspólnej kompilacji nie jest całkowicie dowolny.*

Pomimo tego, że tezy pracy adresują kilka oddzielnych problemów, łącznie, w mojej ocenie, kwalifikują się do badań w ramach rozprawy doktorskiej.

2. Czy Autor rozwiązał postawiony problem i czy użył do tego właściwych metod?

Punktem wyjścia do badań przedstawionych w recenzowanej pracy była dokładna identyfikacja zastanego stanu dziedziny testowania mutacyjnego, uwzględniająca podejmowane w tym zakresie problemy i uzyskane dotąd wyniki. Opis stanu zastanego został przedstawiony w rozdziałach 2, 3 i 4. W ocenie recenzenta jest to wyczerpujące przedstawienie obecnego stanu wiedzy stanowiącej kontekst dla podjętych badań.

W kolejnych rozdziałach (5 – 8) Autor przedstawił podjęte przez siebie problemy oraz uzyskane wyniki badawcze. Każdy z tych problemów został najpierw opisany w ujęciu teoretycznym wraz z naszkicowaniem metody postępowania zmierzającej do jego rozwiązania, a następnie przedstawiono wyniki uzyskane w ramach zastosowania tej metody. Wspólnym mianownikiem zrealizowanych badań jest ich eksperymentalny charakter. Oznacza to, że rozwiązanie każdego z podjętych problemów wymagało zaplanowania odpowiednich eksperymentów, ich realizacji w ramach przyjętych ograniczeń oraz opracowania i interpretacji zebranych danych eksperymentalnych. Co jest również szczególnie istotne, Autor jawnie dokonał refleksji nad wiarygodnością uzyskanych wyników eksperymentalnych rozpatrując różne zagrożenia dotyczące tej wiarygodności.

Powyższe zagadnienia zostały w sposób rzetelny przedstawione w pracy i w mojej ocenie praca nie budzi zastrzeżeń co do zastosowanej metody badawczej.

Realizacja nietrywialnych eksperymentów dotyczących oprogramowania wiąże się zwykle ze znacznymi nakładami pracy oraz najczęściej wymaga opracowania i implementacji odpowiednich środowisk wspomagających. Warte podkreślenia jest, że badania przedstawione w recenzowanej pracy były realizowane z wykorzystaniem takich środków, a Autor pracy był jednym z głównych ich twórców. Środki te zostały opisane w rozdziale 9 oraz w Dodatkach B i C.

Stwierdzam, że do rozwiązania problemów podjętych w pracy Autor zastosował adekwatne metody i techniczne środki wspomagające.

3. Na czym polega oryginalny dorobek Autora i przydatność uzyskanych wyników?

Rozprawa stanowi kilka przyczynków do problemu testowania mutacyjnego, a więc nie jest ona raportem na temat jednego monolitycznego osiągnięcia. Mają one zróżnicowaną „wagę" zarówno jeżeli idzie o wartość merytoryczną jak i zakres udziału Autora w prezentowanych wynikach.

Wspólnym mianownikiem wszystkich wyników jest koncentracja na procesie testowania mutacyjnego z punktu widzenia jego skuteczności i wydajności. Chodzi więc o to, w jaki sposób organizować ten proces, by wywarł on oczekiwany skutek (poprawę jakości testów rozpatrywanego systemu) przy wydatkowaniu mniejszych nakładów (głownie chodzi tu o czas, po którym osiągane są efekty testowania mutacyjnego). Zagadnienie to nie było jak dotąd całkowicie ignorowane (a więc nie jest to problem nowy), ale poświęcono mu mniej uwagi, bardziej koncentrując się na samym doborze mutantów włączanych w zakres procesu testowania. Tak więc prezentowana praca wnosi oryginalny wkład w obszar problemowy, który, chociaż został już wcześniej zidentyfikowany, jak dotąd nie stał się jeszcze przedmiotem dogłębnej penetracji badawczej.

Praca liczy 183 strony i jest podzielona na 10 rozdziałów (w tym wstęp i podsumowanie). Dodatkowo zamieszczono spis literatury, trzy dodatki o charakterze technicznym oraz spisy rysunków, tablic i pseudokodów.

Oryginalne wyniki pracy są zawarte w rozdziałach od 5 do 9. Zawierają one wyniki dotyczące bardziej szczegółowych problemów podjętych przez Autora.

W rozdziale 5 podjęto problem wielu mutantów w jednej kompilacji. Chodzi tu o odpowiedź na pytanie czy wprowadzenie do kodu wielu mutantów i ich wspólna kompilacja, a następnie wybiórcze „aktywowanie" mutantów w ramach poszczególnych testów daje znaczące korzyści wydajnościowe w porównaniu z sytuacją, gdy mutanty są poddawane oddzielnym kompilacjom. Autor zaproponował abstrakcyjny model opisujący sposób wprowadzania wielu mutantów do kodu źródłowego, który następnie jest poddawany jednej kompilacji. Odwołując się do wcześniejszych prac innych autorów wyjaśnił sposób generowania mutantów z wykorzystaniem różnych operatorów mutacyjnych wskazując również, że nie wszystkie mutanty mogą ze sobą współistnieć (Teza 4). Zaproponowany sposób wprowadzania mutantów został następnie zaimplementowany w narzędziu MMIOC, które automatyzuje proces generowania mutantów, ich wspólnej kompilacji oraz selektywnego aktywowania w trakcie testów programu wynikowego. Z wykorzystaniem systemu MMIOC zostały zrealizowane badania eksperymentalne, których celem była ocena uzysku wydajnościowego wynikającego z zastosowania wspólnej kompilacji wielu mutantów. Badania zostały przeprowadzone na pięciu różnych, stosunkowo niedużych programach. Uzyskane wyniki eksperymentalne posłużyły do uzasadnienia Tezy 5.

W rozdziale 6 poszukiwano odpowiedzi na pytanie, czy w sytuacji ewoluującego programu (programu który podlega zmianom) można ograniczyć mutacje jedynie do kodu nowego (lub zmienionego), natomiast nie mutować (ponownie) kodu, który nie ulegał

zmianom. Chociaż ogólnie, ze względu na możliwość dowolnych współzależności pomiędzy elementami programu, nie można wykluczyć że wprowadzona lokalnie zmiana wpłynie na dowolnie „odległy' inny fragment kodu, to pytanie takie jest zasadne i warto je poddać eksperymentalnej ocenie. Uzyskanie pozytywnej odpowiedzi na to pytanie ma potencjalnie duże znaczenie, gdyż zdecydowana większość mającego praktyczne znaczenie oprogramowania podlega ewolucji, a ograniczenie mutacji wyłącznie do zmienionego/nowego kodu może znacznie ograniczyć nakłady na doskonalenie testów takiego oprogramowania (jego testowanie mutacyjne). W szczególności, może to uzasadnić ekonomicznie stosowanie testowania mutacyjnego dla każdej nowej wersji wydawanego oprogramowania. W ramach przeprowadzonego eksperymentu poddano analizie dwa programy *open source*, dla których był możliwy dostęp do ich kolejnych wersji kodu (o rozmiarze w przedziale 12 do 25 KLOC). Wykorzystano również zmodyfikowaną wersję dostępnego narzędzia PIT wspomagającego analizę mutacyjną kodu w języku Java. Wyniki eksperymentu potwierdziły Tezę 1, wykazując że w cyklu ewolucji oprogramowania uzasadnionym jest skoncentrowanie się przede wszystkim na mutacjach kodu nowego lub zmienionego. Jest to wynik o potencjalnie znaczących skutkach praktycznych, gdyż stanowi ważną wskazówkę, gdzie należy koncentrować nakłady na testowanie mutacyjne ewoluującego oprogramowania.

W rozdziale 7 podjęto problem, w jaki sposób spośród znacznej liczby możliwych mutantów potencjalnie generowanych przez operatory mutacyjne (których wyczerpujące przebadanie staje się zbyt kosztowne) wybrać takie, które mają największy potencjał poprawy testów i koncentrując się na nich uzyskać ograniczenie nakładów na testowanie mutacyjne. W szczególności można ten efekt osiągnąć eliminując z procesu testowania mutacyjnego *mutanty trywialne*, tzn. takie które po ich pierwszym wykryciu będą dalej ponownie wykrywane w kolejnych iteracjach testowania mutacyjnego, stając się jedynie kosztem i nie wnosząc wartości dodanej. W swoich badaniach Autor skoncentrował się na identyfikacji *trywialnych operatorów mutacyjnych*, czyli takich których zastosowanie z największym prawdopodobieństwem prowadzi do generowania trywialnych mutantów (warto tu podkreślić, że „trywialność" operatora jest specyficzna względem badanego programu i wyjściowego zestawu testów dla tego programu, a więc niemożliwym jest bezwzględne uszeregowanie operatorów mutacyjnych pod względem ich trywialności). Rozwiązanie zaproponowane przez Autora polega na tym, by decyzję o włączeniu danego mutanta do dalszego procesu testowania mutacyjnego uzależnić od tego, jak często mutant ten był wykrywany w poprzednich iteracjach (a więc w jakim stopniu jest trywialny). Do kwantyfikacji takiego mechanizmu Autor wykorzystał model bayesowski wiążący ze sobą prawdopodobieństwa warunkowe następujących zdarzeń dotyczących danego operatora mutacyjnego: prawdopodobieństwo wylosowania mutanta w poprzednim kroku, przy założeniu że stosunek wykrytych do niewykrytych mutantów związanych z tym operatorem jest znany oraz prawdopodobieństwo wylosowania mutanta w kroku następnym, dla którego stosunek wykrytych do niewykrytych mutantów związanych z tym operatorem nie jest jeszcze znany. Model bayesowski daje podstawę do szacowania drugiego z tych prawdopodobieństw, a więc tworzy warunki do podjęcia decyzji, czy dany mutant będzie uwzględniony w kolejnym kroku. Autor zaproponował i zaimplementował algorytm, który na podstawie kodu badanego programu i związanego z nim zestawu testów podejmuje decyzje o włączaniu w kolejnych krokach mutantów generowanych przez poszczególne operatory mutacyjne. Wykorzystując to narzędzie, zaplanował i zrealizował szereg eksperymentów dla wybranych programów i dla wybranych operatorów mutacyjnych oraz dla różnych zakresów ich stosowania. Wyniki tych eksperymentów stanowią uzasadnienie Tezy 2. Wskazują one na możliwość redukcji liczby mutantów bez znaczących strat w skuteczności procesu testowania mutacyjnego, chociaż dalej występuje możliwość pomijania mutantów „żywych" a więc nie wykrytych. Autor

zaproponował kilka idei dotyczących sposobu zmniejszania wpływu tego efektu, ale potraktował to jako pomysły na dalsze badania i w niewielkim stopniu włączył do bieżącego zakresu. Uzyskane wyniki, chociaż wystarczające do uzasadnienia Tezy 2 (sformułowanej dość ostrożnie) stanowią bardziej punkt wyjścia dla dalszych badań niż w pełni dojrzały wynik badawczy.

W rozdziale 8 Autor podjął problem włączenia testowania mutacyjnego w zwinny proces rozwoju oprogramowania, w szczególności wykorzystujący podejście Test-Driven Development (TDD). W szczególności interesowało go pytanie czy zasadnym jest dodanie do tego procesu komponentu mutacji, a więc przejście od TDD do TDD+M. W tym celu zaproponował miejsce mutacji w modelu TDD, lokując je pomiędzy wykonaniem dotychczasowych testów a następującą po nich refaktoryzacją kodu. Badania służące uzasadnieniu Tezy 3 przeprowadził po jej zdekomponowaniu na trzy bardziej szczegółowe tezy:

Teza 3.1: Testy napisane w podejściu TDD+M dają lepsze pokrycie kodu niż te pisane w czystej metodyce TDD, bez udziału mutacji.

Teza 3.2: Testy napisane w podejściu TDD+M są silniejsze niż te pisane w czystej metodyce TDD.

Teza 3.3: Jakość zewnętrzna oprogramowania jest wyższa, gdy zamiast TDD stosuje się podejście TDD+M.

Taka dekompozycja Tezy 3 jest do zaakceptowania, chociaż oczywiście stanowi zawężenie jej potencjalnych interpretacji. Powyższe tezy pomijają również fragment pierwotnej Tezy 3 głoszący, że „wzbogacenie podejścia Test-Driven Development o testowanie mutacyjne /.../ zwiększa niezależność testowania poprzez obniżenie zjawiska stronniczości (ang. bias) autora testowanego kodu". Warto może by w trakcie obrony pracy Autor ustosunkował się do tej kwestii. Następnie dokonał eksperymentalnej oceny procesów TDD i TDD+M dla wybranej grupy operatorów mutacyjnych. Badania zostały wykonane na równolegle realizowanych procesach wytwórczych dotyczących kilku niewielkich zadań programistycznych. Zadania te były realizowane przez grupy programistów o stosunkowo niedużej dojrzałości zawodowej, ale również włączono w nie programistę o znacznym doświadczeniu zawodowym. Przedmiotem eksperymentów były stosunkowo niewielkie programy poddawane niewielu iteracjom rozwojowym. Wyniki eksperymentu wykazały znaczny wpływ podejścia TDD+M na jakość testów w ramach ewolucji tego samego programu realizowanego przez daną grupę programistów (jakość ta znacząco rosła wraz z kolejnymi iteracjami). Zaskakującym jest jednak, że poprawa ta nie nastąpiła w porównaniu z doświadczonym programistą nie stosującym TDD+M (może to świadczyć o tym, że przewaga kompetencji zawodowej nie jest równoważona przez zastosowanie TDD+M przez zespół niedoświadczonych programistów). Eksperymentami objęto programy numeryczne, a więc dotyczące stosunkowo wąskiej, chociaż ważnej, dziedziny aplikacyjnej. Przeprowadzone eksperymenty zostały starannie zaplanowane, zrealizowane i udokumentowane. Zebrane dane posłużyły do uzasadnienia tez 3.1 i 3.2 na podstawie metryk zebranych w trakcie eksperymentów. Uzasadnienie tezy 3.3., budzi natomiast pewne wątpliwości ponieważ teza ta odnosi się do pojęcia (zewnętrznej) jakości oprogramowania, natomiast w uzasadnieniu skoncentrowano się jedynie na skuteczności testów wynikających z podejścia TDD i TDD+M wykazując, że te ostatnie są bardziej skuteczne (wykrywają więcej defektów). Oczywiście, uwolnienie programu od niektórych z obecnych w nim defektów ma potencjalnie wpływ na wzrost niezawodność tego programu, chociaż w ogólności nie gwarantuje zawsze poprawy niezawodności (na przykład gdy usuwane defekty „maskowały" inne, nie mające wcześniej

wpływu na niezawodność). Pojęcie jakości zewnętrznej jest znacznie szersze niż niezawodność i być może Tezę 3.3 należało sformułować węziej.

Rozdział 9 stanowi uzupełnienie pracy o techniczny opis systemu SAM (Symultanic Automatic Mutation), który został zaprojektowany, zaimplementowany oraz wykorzystany podczas przedstawionych badań. System ten powstał w zespole badawczym, którego Autor jest uczestnikiem. Wkład Autora w zaprojektowanie i w realizację tego systemu był znaczący. Tak więc w zakresie recenzowanej pracy mieści się również nietrywialny wynik związany z budową infrastruktury badawczej, która może być wykorzystana do dalszych badań

Podsumowując uważam, że w recenzowanej pracy podjęto oryginalne problemy badawcze dotyczące ważnych zagadnień mających znaczenie dla jakości oprogramowania. Sformułowano tezy, które zostały następnie uszczegółowione i uzasadnione z wykorzystaniem adekwatnych metod badawczych. Na szczególne podkreślenie zasługuje zaplanowanie i zrealizowanie badań eksperymentalnych, a więc ukierunkowanie Autora na pozyskanie i interpretację danych pochodzących z procesów programistycznych zbliżonych do rzeczywistych. Praca ma charakter teoretyczno-eksperymentalny. Zawarte w niej tezy oraz stopień ich walidacji powodują, że uzasadnionym wydaje się być podjęcie prób zastosowania praktycznego uzyskanych wyników, co jest niewątpliwa zaletą tej pracy.

Wyniki przedstawione w pracy zostały już częściowo opublikowane z udziałem Autora rozprawy. W spisie literatury zamieszczone dwie takie pozycje: *Michał Mnich, Adam Roman, and Piotr Wawrzyniak, Mutation churn model, Schedae Informaticae (2017)*, IF=0,167 wg Academic Accelerator oraz *Adam Roman, Michał Mnich, Test-driven development with mutation testing – an experimental study, Software Quality Journal (2020)*, IF=1.46 wg Academic Accelerator.

4. Czy rozprawa świadczy o dostatecznej wiedzy Autora i znajomości współczesnej literatury?

Autor bardzo dobrze orientuje się w dziedzinie testowania oprogramowania, a w szczególności w zakresie testowania mutacyjnego, zarówno w wymiarze teoretycznym jak i praktycznym. Znalazło to również odzwierciedlenie w spisie źródeł zamieszczonym w pracy.

Autor zademonstrował również wysoką kompetencję w zakresie badań eksperymentalnych, planowaniu eksperymentów, przygotowaniu środków technicznych do ich realizacji oraz opracowaniu i interpretacji uzyskiwanych wyników. Dobrze to rokuje dla jego dalszych prac badawczych, gdyż w dziedzinie inżynierii oprogramowania badania eksperymentalne stanowią bardzo ważny środek do poszerzania zakresu wiedzy i potwierdzania jej praktycznej przydatności.

5. Uwagi krytyczne

W trakcie lektury recenzowanej pracy pojawiło się kilka refleksji ogólnych i uwag szczegółowych, niektóre z nich zamieszczono poniżej.

1. Rozprawa stanowi kilka przyczynków do problemu testowania mutacyjnego, a więc nie jest raportem na temat jednego monolitycznego osiągnięcia. Mają one zróżnicowaną „wagę" zarówno jeżeli idzie o wartość merytoryczną jak i zakres udziału Autora w prezentowanych wynikach. Powstaje pytanie, czy można było sformułować wspólną tezę na bardziej ogólnym poziomie, a dopiero potem dokonać jej dekompozycji na tezy bardziej szczegółowe.
2. Kolejność rozdziałów pracy nie jest zgodna z kolejnością wyszczególnienia tez w rozdziale 1. Czy jest to efekt zamierzony, innymi słowy czy był jakiś zamiar w omawianiu tez w kolejności 4, 5, 1, 2, 3?

3. Pewnym niedostatkiem redakcyjnym jest brak jawnego przewodnika po publikacjach Autora z wskazaniem na to, które wyniki i gdzie zostały już opublikowane.
4. Odnoszę wrażenie, że mówiąc o „efektywności" procesu testowania mutacyjnego w rzeczywistości Autor ma czasem na myśli wydajność tego procesu. Ogólnie, pojęcia anglojęzyczne *effectiveness* i *efficiency* należałoby tłumaczyć jako efektywność/skuteczność (a więc zdolność do osiągania zamierzonego efektu) oraz wydajność (a więc poziom konsumpcji zasobów niezbędnych do osiągnięcia zamierzonego efektu).
    a. Na przykład, na stronie 64 przy opisie celów Eksperymentu (sekcja 5.4) jest mowa o tym, że chodziło o 'sprawdzenie efektywności procesu testowania mutacyjnego' na drodze porównania odpowiednich czasów realizacji elementów tego procesu. A więc w rzeczywistości chodziło tu o wydajność procesu a nie o jego efektywność. Badanie efektywności oznaczałoby tu raczej skupienie się na zdolności badanego procesu do wykrywania („zabijania") wprowadzonych mutantów.
5. Wprowadzenie wielu mutantów do kodu źródłowego (z wykorzystaniem mechanizmu kompilacji warunkowej) ma najpewniej wpływ na czas kompilacji (w porównaniu do sytuacji, gdzie kompilacja dotyczy pojedynczego mutanta). Czy to potencjalne wydłużenie czasu kompilacji zostało uwzględnione w opracowaniu wyników?
6. Wnioski z eksperymentów nie zawsze zostały wystarczająco uzasadnione. Na przykład w sekcji „7.4.2 Wnioski z eksperymentu" pojawia się stwierdzenie, że „problem (znaczącej wartości współczynnika utraconych danych związanych z niewykrytymi mutantami) będzie zanikał z czasem kolejnych sesji mutacyjnych" jednak nie znalazłem uzasadnienia tego stwierdzenia.
7. Każdy program jest eksploatowany w swoim środowisku docelowym, a znajomość tego środowiska, wyrażona *profilem operacyjnym* programu może być bardzo istotną wskazówką dla doboru testów: należy dobierać przypadki testowe mieszczące się w profilu operacyjnym, natomiast można ignorować (dopóki nie nastąpi zmiana profilu operacyjnego) przypadki testowe wykraczające poza ten profil. Testowanie mutacyjne, a w szczególności operatory umożliwiające dobór mutantów raczej ignorują ten aspekt. Czy Doktorant widzi możliwość praktycznej realizacji *testowania mutacyjnego w profilu operacyjnym* umożliwiającego selekcję jedynie tych mutantów, które nie wyprowadzają poza profil operacyjny programu.
8. Podobnie, można zadać pytanie, czy i w jaki sposób można wykorzystać wiedzę o ryzyku związanym z użyciem programu w jego środowisku docelowym w celu bardziej selektywnego doboru mutantów w ramach *testowania mutacyjnego uwarunkowanego ryzykiem.*
9. Zrealizowane eksperymenty, chociaż z pewnością nietrywialne, odbiegają jednak od rzeczywistości przemysłowej. Czy doktorant widzi, w perspektywie dalszych badań, drogę do potwierdzenia czy uzyskane wyniki są skalowalne do w pełni realnych procesów programistycznych.
10. Uwagi redakcyjne (wybrane przykłady):
    a. Str. 47, pierwszy akapit umieszczony pod tabelą 4.1 odwołuje się do numerów kolumn (4-7) jednak w tabeli nie zamieszczono jawnie takiej numeracji
    b. Str. 62, pierwszy akapit pod rysunkiem 5.3: drugie odwołanie do listingu 5.11 jest najpewniej błędne, gdyż chodzi o listing 5.12
    c. Definicja 5.2 na stronie 66 wprowadza pojęcie złożoności cyklomatycznej (CC) natomiast na stronie 77 CC jest zdefiniowana inaczej. Również niektóre definicje są

wprowadzane w ramkach (np. ta na stronie 66) podczas gdy inne bez takiego wyróżnienia. Nie do końca rozumiem zasadę rządzącą takim rozróżnianiem.

d. Str. 86, trzeci i czwarty wiersz trzeciego akapitu: jest: „W kolejnym kroku wartość ta staje się wartością a priori", powinno (chyba) być: „W kolejnym kroku wartość ta staje się wartością a posteriori"

e. Str. 91. Początek tekstu w sekcji 7.4.1 odwołuje się do Tabeli 7.4.1. Jednak tabeli o takim numerze nie ma w pracy.

Podsumowanie

Biorąc pod uwagę zarówno silne strony ocenianej rozprawy jak i dotyczące jej uwagi krytyczne stwierdzam, że zadawalająco spełnia ona wymogi obowiązującej Ustawy i wnioskuję o jej dopuszczenie do publicznej obrony.

Gdańsk 1.09.2021