Bartosz Walter

Autoreferat

przedstawiający opis dorobku i osiągnięć naukowych

załącznik nr 2
do wniosku o wszczęcie postępowania habilitacyjnego

1 Informacje o wykształceniu i przebiegu zatrudnienia

a. Imię i nazwisko: Bartosz Walter
b. Data i miejsce urodzenia: 28 kwietnia 1976 roku w Poznaniu
c. Przebieg pracy zawodowej i naukowej:
    i. 2000-2004 zatrudniony na stanowisku asystenta w Instytucie Informatyki Politechniki Poznańskiej
    ii. 2004-2016 zatrudniony na stanowisku adiunkta w Instytucie Informatyki Politechniki Poznańskiej
    iii. od 2016 zatrudniony na stanowisku starszego wykładowcy w Instytucie Informatyki Politechniki Poznańskiej (od 2018 w niepełnym wymiarze czasu)
    iv. od 2018 r zatrudniony w Poznańskim Centrum Superkomputerowo-Sieciowym (afiliowanym przy Instytucie Chemii Bioorganicznej PAN w Poznaniu)
d. Wykształcenie
    i. 2000 – dyplom magistra inżyniera informatyka w specjalności Inżynieria oprogramowania (z wyróżnieniem)
    ii. 2004 – stopień doktora nauk technicznych w dyscyplinie informatyka, w specjalności inżynieria oprogramowania (z wyróżnieniem), nadany przez Radę Wydziału Informatyki i Zarządzania Politechniki Poznańskiej, na podstawie rozprawy pt. Analiza przekształceń refaktoryzacyjnych (promotor: dr hab. inż. J. Nawrocki, prof. nadzw.)
e. Odbyte szkolenia i posiadane certyfikaty
    i. ITIL (Information Technology Infrastructure Library): Foundation Level
    ii. ISTQB (International Software Testing Qualification Board): Foundation Level, Agile Tester Extension, Full Advanced Level, Test Automation Engineer
f. Znajomość języków obcych:
    i. Angielski: płynnie w mowie i piśmie (poziom C1)
    ii. Niemiecki: podstawowa (poziom A2)

2 Osiągnięcie naukowe stanowiące podstawę wniosku o wszczęcie postępowania habilitacyjnego:

Do dorobku wchodzącego w skład osiągnięcia naukowego stanowiącego podstawę złożenia wniosku habilitacyjnego wchodzą następujące publikacje, składające się na monotematyczny cykl prac pt.

Code smells as early predictors of source code maintainability. Detection approaches, relationships and impact on selected code properties

[A1] **Bartosz Walter**, Błażej Pietrzak, *Multi-criteria Detection of Bad Smells in Code with UTA Method*, Proc. of XP 2005 Conference, Springer, LNCS Vol. 3556, pp. 154-161 (**MNiSW: 13**)

[A2] Błażej Pietrzak, **Bartosz Walter**, *Leveraging Code Smell Detection with Inter-smell Relations*, Proc. of XP 2006 Conference, Springer, LNCS Vol. 4044, pp: 75-84 (**MNiSW: 13**)

[A3] Francesca Arcelli Fontana, Vincenzo Ferme, Alessandro Marino, **Bartosz Walter**, Paweł Martenka, *Investigating the Impact of Code Smells on System's Quality: An Empirical Study on Systems of Different Application Domains*, ICSM 2013: 260-269 (**MNiSW: 15**)

[A4] Aiko Yamashita, Marco Zanoni, Francesca Arcelli Fontana, **Bartosz Walter**, *Inter-smell relations in industrial and open source systems: A replication and comparative analysis*, 2015 IEEE International Conference on Software Maintenance and Evolution (ICSME), IEEE, pp: 121-130. (**MNiSW: 15**)

[A5] Francesca Arcelli Fontana, Jens Dietrich, **Bartosz Walter**, Aiko Yamashita, Marco Zanoni, *Antipattern and Code Smell False Positives: Preliminary Conceptualization and Classification*, SANER 2016: 609-613 (**MNiSW: 15**)

[A6] **Bartosz Walter**, Tarek Alkhaeir, *The relationship between design patterns and code smells: An exploratory study.* Information and Software Technology 74 (2016), pp. 127-142 (**MNiSW: 35**)

[A7] **Bartosz Walter**, Francesca Arcelli Fontana, Vincenzo Ferme: *Code smells and their collocations. A large-scale experiment on open-source systems,* Journal of Systems and Software 144 (2018), pp. 1-21 (2018) (**MNiSW: 35**)

2.1 Wprowadzenie

Pielęgnacja odgrywa kluczową rolę w cyklu rozwojowym oprogramowania. Badania wskazują, że relatywny koszt przypadający na tę fazę w odniesieniu do całkowitego kosztu związanego z funkcjonowaniem programu wynosi nawet 80% i wykazuje tendencję wzrostową. Jedną z przyczyn takiego stanu rzeczy jest narastające nieuporządkowanie w kodzie programu, mierzone przez tzw. dług technologiczny (ang. *technical debt*). Odzwierciedla on ilość pracy, którą trzeba włożyć w usunięcie negatywnych skutków ubocznych wprowadzonych wcześniej zmian w oprogramowaniu: modyfikacji, poprawek czy restrukturyzacji. Dług technologiczny nie ma jednak jednoznacznie negatywnego wpływu na pielęgnowalność oprogramowania: z jednej strony pozwala na odsunięcie w czasie pewnych prac pielęgnacyjnych, ale z drugiej powoduje konieczność ponoszenia wyższych kosztów tej pielęgnacji.

W związku z tym ograniczenie kosztu pielęgnacji oprogramowania, stanowiące jedno z wyzwań współczesnej inżynierii oprogramowania, wymaga m.in. przyjęcia i wdrożenia skutecznej strategii zarządzania długiem technologicznym. Jednym z jej elementów jest możliwie wczesne identyfikowanie czynników, które ten dług mogą zwiększać. Ma to szczególne znaczenie w dobie upowszechnienia tzw. zwinnych metodyk wytwarzania oprogramowania, zgodnie z którymi system informatyczny jest nieustannie modyfikowany i rozwijany, przez co znajduje się w fazie pielęgnacji przez większą część swojego życia.

Z tego powodu moje zainteresowania naukowe po uzyskaniu stopnia doktora nauk technicznych skierowałem właśnie w stronę metod rozpoznawania symptomów problemów dotyczących pielęgnowalności oprogramowania. W szczególności, skupiłem uwagę na tzw. *przykrych zapachach* (ang. *code smells*) znajdujących się w kodzie programu. Są to stosunkowo łatwe w identyfikacji, powtarzające się wzorce znajdujące się w analizowanym programie, wskazujące na nieoptymalne rozwiązania projektowe i implementacyjne, mogące być w przyszłości źródłem długu technologicznego. Przykre zapachy w kodzie programu mogą pełnić rolę predyktorów dla praktycznych problemów związanych z pielęgnacją, np. nadmiernej złożoności, błędnej modularyzacji oprogramowania lub niepoprawnego wykorzystania niektórych mechanizmów obecnych w językach oprogramowania. Z drugiej strony należy pamiętać, że przykre zapachy są jedynie symptomami, które mogą, ale nie muszą wskazywać na pojawienie się problemów. Oznacza to, że związek między ich obecnością a spodziewanym pogorszeniem się pielęgnowalności programu jest obarczony niepewnością, a co za tym idzie – może być uwarunkowany wieloma czynnikami kontekstowymi i subiektywnymi.

Ważną motywacją dla podjęcia w mojej pracy tematyki przykrych zapachów i ich znaczenia w praktyce wytwarzania oprogramowania jest popularność, jaką ta metafora cieszy się w przemyśle. Dostępność narzędzi do detekcji zapachów oraz stosowania przekształceń refaktoryzacyjnych powoduje, że idea tworzenia tzw. *czystego kodu,* czyli kodu przejrzystego, charakteryzującego się możliwie niskim poziomem długu technologicznego, jest obecnie powszechnie stosowana w praktyce. Nadal jednak trwa debata dotycząca wielu aspektów związanych z obecnością przykrych zapachów,

dotyczących m.in. kwantyfikowalnej oceny ich wpływu na ewolucję oprogramowania czy czynników, które mogą na nie wpływać.

Dlatego główny nurt moich badań dotyczył właśnie kilku obszarów związanych z wykrywaniem przykrych zapachów, wzajemnymi relacjami między nimi oraz ich związkiem z pielęgnowalnością programów. W szczególności, skupiłem się na czterech najważniejszych kierunkach, powiązanych wzajemnie ze sobą:

1. Identyfikacji nowych źródeł informacji na temat obecności przykrych zapachów oraz ocenie możliwości zastosowania wielokryterialnych metod ich łączenia w celu poprawy dokładności wykrywania przykrych zapachów.
2. Identyfikacji typów relacji, w jakich mogą występować różne przykre zapachy, oraz badaniu ich wpływu na ważne praktyczne właściwości oprogramowania.
3. Identyfikacji wybranych czynników kontekstowych na obecność przykrych zapachów.
4. Analizie fałszywie pozytywnych przypadków przykrych zapachów.

Wyniki badań dotyczących tych czterech kierunków przedstawiłem w kolejnych podrozdziałach.

2.2 Identyfikacja nowych źródeł informacji na temat obecności przykrych zapachów oraz ocena możliwości zastosowania wielokryterialnych metod ich łączenia w celu poprawy dokładności wykrywania przykrych zapachów.

Tradycyjne modele oceny pielęgnowalności kodu, a w konsekwencji opracowania strategii refaktoryzacji tego kodu były oparte przede wszystkim na wartościach statycznych metryk kodu dotyczących wybranej własności analizowanego programu (Simon et al., 2001), np. jego rozmiaru, złożoności, głębokości drzewa dziedziczenia klas lub stopnia sprzężenia pomiędzy modułami. W dalszej kolejności, na ich podstawie powstały bardziej złożone metody wykrywania przykrych zapachów polegające na połączeniu za pomocą funkcji logicznej (zwykle koniunkcji lub alternatywy) kilku metryk związanych z różnymi własnościami kodu istotnymi dla danego zapachu. Metody te zyskały sporą popularność dzięki wysokiej dokładności wyników i przejrzystości procesu analizy, jednak nie uwzględniały one źródeł informacji innych niż statyczne metryki.

Dlatego w pracy [A1] zaproponowano bardziej holistyczną perspektywę przykrych zapachów poprzez wskazanie nowych symptomów i ich źródeł, które mogą mieć wpływ na obecność tych anomalii.

W wyniku przeprowadzonej analizy zidentyfikowano sześć podstawowych źródeł, które mogą dostarczyć informacji dotyczącej obecności różnych przykrych zapachów. Są to:

- **Doświadczenie i intuicja ekspercka**, pozwalające nieformalnie połączyć wiele czynników w subiektywną ocenę obecności przykrego zapachu w analizowanym kodzie programu;

- **Metryki dotyczące statycznych własności kodu**, np. jego rozmiaru, złożoności, stopnia sprzężenia, lub stopnia wykorzystania określonych mechanizmów udostępnianych przez język programowania, np. dziedziczenia, polimorfizmu czy hermetyzacji;
- **Informacje dotyczące strukturalnych własności kodu**, np. zależności pomiędzy poszczególnymi modułami, obecności określonych struktur czy użyciu charakterystycznych konstrukcji języka programowania;
- **Historia poszczególnych modułów lub klas**, zapisana w repozytoriach systemów zarządzania zmianami (np. Git, Mercurial SCM czy SVN);
- **Analiza zachowania programu**, np. wyniki wykonania testów jednostkowych i integracyjnych, uruchomionych dla określonych danych wejściowych;
- **Relacje pomiędzy różnymi przykrymi zapachami** obecnymi w obrębie tej samej klasy (modułu) lub klas (modułów) powiązanych różnymi relacjami.

Tradycyjnie, pierwsze dwa źródła informacji pełniły dominującą rolę w procesie wykrywania przykrych zapachów. Rozwój automatycznych metod wykrywania przykrych zapachów doprowadził do powstania rozwiązań agregujących wiele metryk w celu odzwierciedlenia typowych elementów charakterystycznych dla danego zapachu, np. strategii wykrywających (ang. *detection strategies*) (Lanza et al., 2005) czy DECOR (Moha et al., 2009).

W modelu zaproponowanym w analizowanej pracy podstawowym źródłem informacji o obecności przykrego zapachu pozostają metryki, jednak można je efektywnie uzupełnić o dane pochodzące z innych źródeł. Przykłady podane w tej pracy oraz [B11] wskazują, że takie połączenie źródeł informacji pozwala poprawić dokładność identyfikacji, uwzględnić nowe objawy związane z danym przykrym zapachem lub ograniczyć złożoność procesu identyfikacji.

Ponadto, niektóre ze źródeł informacji o przykrych zapachach zidentyfikowanych w tej pracy stały się przedmiotem niezależnych dalszych badań, np. metoda wykrywania przykrych zapachów w oparciu o nadzorowane przez ekspertów uczenie maszynowe (Arcelli et al., 2016; Nucci et al., 2018), analiza repozytoriów kodu (Palomba et al, 2016) czy analiza związków pomiędzy przykrymi zapachami (Palomba et al., 2018; A7)

Jednoczesna ocena danych pochodzących z wielu źródeł wymaga jednak zastosowania odpowiedniej metody łączącej te informacje. W pracy [A1] przedstawiono możliwość wykorzystania metody UTA (UTilites Additives), która uwzględnia wielokryterialny obraz analizowanego zjawiska w celu stworzenia rankingu poszczególnych instancji modułów, w których obecne są poszczególne objawy przykrego zapachu, uwzględniającego przy okazji model preferencji użytkownika odzwierciedlający ich wpływ na dokonaną ocenę. W tym przypadku analizie poddano przykry zapach *Large Class*, a za źródło kodu posłużył projekt Apache Tomcat 5.5.4. Zapach ten posiada kilka różnych cech charakterystycznych, które można identyfikować i mierzyć za pomocą metryk, ale także analizując w tym celu strukturę badanej klasy oraz wykrywając związki z innymi przykrymi zapachami. Wyniki przedstawione w pracy wskazują, że

wielokryterialna analiza obecności przykrych zapachów, oparta także na innych niż metryki źródłach danych, pozwala z wysoką skutecznością odwzorować model preferencji użytkownika. To z kolei umożliwia tworzenie metod detekcji dostosowanych do poszczególnych użytkowników i ich percepcji długu technologicznego, minimalizując liczbę przypadków fałszywie pozytywnych, a także zwiększając precyzję szacowania długu technologicznego związanego z obecnością poszczególnych przykrych zapachów.

2.3 Identyfikacja typów relacji, w jakich mogą występować różne przykre zapachy, oraz badanie ich wpływu na ważne praktyczne właściwości oprogramowania.

Możliwość występowania związków pomiędzy różnymi przykrymi zapachami w obrębie tej samej klasy została zasugerowana w książce Fowlera (Fowler, 2001), a następnie wskazana jako źródło informacji, dzięki któremu można poprawić dokładność procesu wykrywania przykrych zapachów [A1]. Kwestia ta została następnie rozwinięta w pracy [A2], w której zaproponowano taksonomię rodzajów związków pomiędzy przykrymi zapachami obecnymi w tej samej klasie, oraz podano przykłady ich praktycznego wykorzystania w procesie detekcji. Zidentyfikowano 7 rodzajów relacji, jakie mogą wystąpić pomiędzy przykrymi zapachami A i B:

- *Wsparcie proste*, w której obecność przykrego zapachu A zwiększa prawdopodobieństwo obecności zapachu B;
- *Wsparcie złożone*, w której jednoczesna obecność kilku przykrych zapachów A_1, A_2, …, A_n zwiększa prawdopodobieństwo obecności zapachu B;
- *Wsparcie przechodnie*, w której obecność przykrego zapachu A zwiększa prawdopodobieństwo wystąpienia zapachu B, który z kolei zwiększa prawdopodobieństwo dla zapachu C;
- *Wykluczenie*, w której obecność przykrego zapachu A redukuje (lub nawet usuwa) prawdopodobieństwo obecność przykrego zapachu B. Przykład: Klasa o nadmiernej odpowiedzialności (*God Class*) nie może być jednocześnie klasą zubożałą (*Lazy Class*), dlatego te dwa zapachy się wzajemnie wykluczają;
- *Zawieranie*, w której przykry zapach A jest szczególnym przypadkiem przykrego zapachu B;
- *Wsparcie wzajemne*, stanowiącą symetryczne domknięcie relacji wsparcia prostego: obecność dowolnego z zapachów A i B zwiększa prawdopodobieństwo wystąpienia drugiego z nich;
- *Wspólne przekształcenie refaktoryzacyjne* usuwające oba zapachy A i B.

Relacje wyróżnione w tej taksonomii mogą dotyczyć zarówno przykrych zapachów w obrębie pojedynczej jednostki leksykalnej (zwykle klasy), jak i jednostek powiązanych relacją zależności. Na przykład, klasa obciążona przykrym zapachem *Data Class*, czyli stanowiąca jedynie kontener do przechowywania danych, ale pozbawiona metod pozwalających na wykorzystanie tych danych, będzie stanowić źródło danych dla innej klasy, która będzie odpowiedzialna za ich przetwarzanie. Sytuacja taka wyczerpuje znamiona innego przykrego zapachu, *Feature Envy*, w związku z czym zapachy te, obecne w zależnych klasach, także są ze sobą powiązane relacją. Można je nazwać

sprzężonymi (ang. *coupled smells*) ponieważ dotyczą one dwóch różnych klas, pomiędzy którymi występuje relacja sprzężenia.

Natomiast przykładem *współwystępujących przykrych zapachów* (ang. *collocated smells*) może być rozbudowana hierarchia wyrażeń warunkowych umieszczona wewnątrz metody (stanowiąca przykry zapach *Switch Statements*), powodująca jednocześnie nadmierny wzrost złożoności przetwarzania danych w tej metodzie (opisywanego przez przykry zapach *God Method*). Oba przykre zapachy znajdują się wewnątrz tej samej jednostki leksykalnej w kodzie.

Znajomość relacji pomiędzy przykrymi zapachami w wielu przypadkach pozwala na dokładniejsze lokalizowanie obu przykrych zapachów, a także zastosowanie strategii refaktoryzacyjnej, która pozwoli usunąć oba zapachy. Przedstawiona klasyfikacja rodzajów relacji wyznacza zatem nowe kierunki badań nad przykrymi zapachami i umożliwia ich wykorzystanie w procesie ich wielokryterialnej identyfikacji.

Rozwijając ten wątek, w pracy [A4] przedstawiono wyniki badań nad trzema systemami: dwoma z udostępnionym i jednym z zamkniętym kodem źródłowym, rozwijanego w warunkach przemysłowych, także dotyczących występowania relacji pomiędzy przykrymi zapachami.

Analizie poddano 15 rodzajów przykrych zapachów oraz oba omawiane typy relacji pomiędzy nimi: *współwystępowanie* oraz *sprzężenie*. W pracy tej przyjęto szeroką definicję zależności między klasami X i Y: zgodnie z nią, X zależy od Y, jeżeli do prawidłowego funkcjonowania klasy X konieczna jest obecność klasy Y (a zatem uwzględnia ona dziedziczenie po klasie Y, odwołanie do atrybutu klasy Y, utworzenie obiektu klasy U, posiadanie przez X zmiennej typu Y, wywołanie w klasie X metody zadeklarowanej w klasie Y, ora obecność w metodzie klasy X wartości zwracanej lub parametru typu Y).

Badanie przeprowadzono z użyciem analizy głównych składowych (PCA), osobno dla zapachów współwystępujących i sprzężonych.

Uzyskane wyniki wskazują, że niektóre związki pomiędzy przykrymi zapachami występują we wszystkich badanych systemach, podczas gdy inne zostały zidentyfikowane tylko w części systemów, przy czym obserwacja ta dotyczy zarówno relacji współwystępowania, jak i sprzężenia. Może to wskazywać, że relacje między przykrymi zapachami co do zasady nie dotyczą tylko jednego typu związku, jednak konkretne zapachy mogą wchodzić w różne zależności, zależne od rodzaju relacji pomiędzy klasami, których dotyczą.

Drugim ważnym wynikiem przedstawionym w pracy [A4] było eksperymentalne potwierdzenie sugerowanych w innych pracach relacji współwystępowania pomiędzy określonymi typami przykrych zapachów, np. {*God Class; Feature Envy*} czy {*God Class*; *Intensive Coupling*}, a także relacji sprzężenia, np. {*Data Class*; *Feature Envy*}.

Ponadto, w badanych systemach stwierdzono obecność nowego podtypu relacji sprzężenia między przykrymi zapachami: *komponentów redundantnych*, w których ten sam rodzaj zapachu występuje w klasach sprzężonych ze sobą. Dotyczy to zapachów *Intensive Coupling, Message Chains, External* i *Sibling Duplication, Data Clumps, Blob Operation* i *Refused Parent Bequest.*

Jednym z ważniejszych wniosków przedstawionych w tej pracy jest możliwość powiązania analizy współwystępowania i sprzężenia przykrych zapachów w celu ograniczenia liczby przypadków fałszywie pozytywnych, które, mimo pozornego spełnienia kryteriów wystąpienia anomalii, w rzeczywistości nie mają szkodliwego wpływu na pielęgnowalność analizowanego systemu. Praca ta wskazuje również na potrzebę bliższej analizy czynników kontekstowych, które mogą w znacznym stopniu wpływać na obecność zarówno pojedynczych przykrych zapachów, jak i relacji między nimi. Ich uwzględnienie powinno pozwolić ograniczyć liczbę fałszywie pozytywnych przypadków takich zapachów i podnieść dokładność używanych obecnie narzędzi.

Uwzględnienie we wzmiankowanej pracy dwóch różnych środowisk, w jakich zwykle powstaje oprogramowanie (tj. z udostępnionym i zamkniętym źródłem) pozwoliło także na stwierdzenie, czy środowisko może stanowić czynnik różnicujący w przypadku relacji między przykrymi zapachami.

Wątek analizy typowych relacji występujących pomiędzy przykrymi zapachami kontynuowano w pracy [A7]. Przedstawiono w niej wyniki analizy kodu źródłowego 68 projektów w języku Java, pochodzącego z popularnego zbioru Qualitas Corpus (Tempero et al.., 2010), przeprowadzonej dla przykrych zapachów współwystępujących w tych samych klasach. Motywacją do podjęcia badań empirycznych nad związkami między przykrymi zapachami na dużej próbie danych była obserwowana w literaturze znacząca nierównowaga pomiędzy relacjami sugerowanymi na podstawie anegdotycznych obserwacji a danymi eksperymentalnymi, które wspierałyby te wyniki.

W pracy przeanalizowano 14 przykrych zapachów, do których detekcji użyto w sumie 7 narzędzi. Praca ta jest, jak dotąd, największą ilościową analizą zjawiska współwystępowania przykrych zapachów w programach w języku Java.

Do najważniejszych wyników przedstawionych w tym artykule należą:

- empiryczna walidacja wielu związków między przykrymi zapachami, które były podawane przez różnych autorów, jednak bez poparcia znaczącymi danymi ilościowymi;
- zidentyfikowanie nowych, wcześniej nieznanych relacji pomiędzy przykrymi zapachami.

Ponadto, w pracy tej udzielono odpowiedzi na następujące pytania dodatkowe:

Q1. W jaki sposób zastosowana metoda analizy związków między przykrymi zapachami wpływa na wyniki tej analizy?

Aby udzielić odpowiedzi na to pytanie, analizę zebranego materiału przeprowadzono na trzy sposoby:

- Obliczając współczynnik korelacji Spearmana dla par współwystępujących przykrych zapachów;
- Stosując analizę głównych składowych (PCA, ang. *Principal Component Analysis*) jako metodę eksploracji współwystępowania większych grup przykrych zapachów;
- Dokonując ekstrakcji reguł asocjacyjnych w zbiorze transakcji zbudowanym z klas posiadających przynajmniej dwa przykre zapachy.

Wyniki uzyskane za pomocą dwóch pierwszych metod są w dużej mierze zbieżne, natomiast zastosowanie reguł asocjacyjnych pozwoliło odnaleźć inne związki między przykrymi zapachami. Wskazuje to na potrzebę komplementarnego stosowania obu metod w celu całościowej oceny zjawiska.

Q2. Czy zastosowanie większej liczby detektorów przykrych zapachów wpływa na wyniki analiz relacji między tymi zapachami?

Aby odpowiedzieć na to pytanie, analizę z wykorzystaniem wymienionych wcześniej metod przeprowadzono osobno dla zbiorów danych reprezentujących trzy poziomy wiarygodności: 25%, 50% i 75%, oznaczających zgodność wyników działania detektorów użytych do identyfikacji danego zapachu. Uzyskane wyniki pokazały, że część zapachów jest wykrywana przez większość detektorów, podczas gdy inne wykazują duże zróżnicowanie. Może to stanowić wskazówkę dotyczącą wykorzystania najbardziej wiarygodnych narzędzi w procesie identyfikacji zapachów.

Q3. Czy zidentyfikowane relacji między przykrymi zapachami zależą od dziedziny programu?

Analizę przeprowadzono osobno dla zbiorów zbudowanych z projektów zakwalifikowanych do różnych dziedzin zastosowań, zgodnie z ich podziałem zaproponowanym w pracy [A3]. W efekcie, poza nielicznymi wyjątkami, nie stwierdzono istotnych różnic pomiędzy dziedzinami, co oznacza, że nie stanowią one czynnika kontekstowego różnicującego relację współwystępowania między przykrymi zapachami.

2.4 Identyfikacja wybranych czynników kontekstowych wpływających na obecność przykrych zapachów.

Kontynuując rozważania dotyczące czynników kontekstowych, które mogą wpływać na obecność przykrych zapachów, w pracy [A6] przedstawiono wyniki eksploracyjnej oceny interakcji pomiędzy wzorcami projektowymi oraz przykrymi zapachami współwystępującymi w tych samych klasach. Wzorce projektowe (Gamma et al., 1995) są celowo stosowanymi rozwiązaniami, które zostały poddane szczegółowej analizie w

określonym kontekście, w wyniku której zidentyfikowano konsekwencje ich użycia. Natomiast przykre zapachy są symptomami problemów dotyczących pielęgnowalności, które są niezamierzonym efektem ubocznym zmian wprowadzonych w kodzie. W związku z tym interakcje zachodzące pomiędzy nimi mogą także wpływać na sposób, w jaki kod ewoluuje, np. poprzez ograniczenie pozytywnego wpływu wzorców projektowych lub wywołanie nowych efektów ubocznych.

Celem analizy było określenie, czy klasy, w których współwystępują wzorce projektowe i przykre zapachy, zachowują się inaczej niż pozostałe klasy. W tym celu badaniu poddano 2 systemy z otwartym kodem źródłowym, porównując między sobą cztery zbiory danych zawierające możliwe konfiguracje dotyczące obecności i nieobecności wzorców projektowych i przykrych zapachów.

Uzyskane wyniki wskazują, że klasy będące elementami wzorców projektowych są istotnie rzadziej obarczone przykrymi zapachami niż pozostałe klasy, a udział klasy we wzorcu jest negatywnym predyktorem obecności w niej przykrego zapachu. Oznacza to, że udział we wzorcu projektowym może stanowić czynnik kontekstowy dla obecności przykrych zapachów. Ponadto, wśród klas z obecnym przykrym zapachem stosunek liczby klas nieuczestniczących we żadnym wzorcu do liczby klas pełniących rolę we wzorcu pozostaje stabilny wraz z ewolucją danego systemu lub lekko maleje.

Przeprowadzona analiza z wykorzystaniem reguł asocjacyjnych pozwoliła znaleźć prawidłowości w relacjach wzorców i przykrych zapachów. Żaden wzorzec nie wykazuje pozytywnej korelacji z jakimkolwiek przykrym zapachem. Do wzorców najrzadziej obciążonych przykrymi zapachami należą *Singleton*, *State*, *Strategy*, *Adapter*, *Command* i *Factory Method*. Na szczególną uwagę zasługuje obecność w tej grupie wzorca *Singleton*, który tradycyjnie uważany jest za przykład nieobiektowego stylu programowania, posiadającego wiele negatywnych konsekwencji dla pielęgnacji programów. Wyniki wskazują jednak, że wzorzec ten stosunkowo rzadko jest obarczony przykrymi zapachami, co ponownie może sugerować istnienie czynników kontekstowych, dotyczących tym razem związków wzorców i przykrych zapachów. Wyniki przedstawione w tej pracy zostały następnie poddane replikacji i pozytywnie zweryfikowane przez innych autorów.

2.5 Analiza fałszywie pozytywnych przypadków przykrych zapachów.

Ważnym elementem pozwalającym na oszacowanie kosztu przyszłej pielęgnacji oprogramowania jest znajomość rozkładu występowania poszczególnych rodzajów przykrych zapachów w różnych programach. W pracy [A3] przedstawiono wyniki analizy przeprowadzonej na 68 systemach z udostępnionym kodem źródłowym, pochodzących ze zbioru Qualitas Corpus. Systemy te zostały pogrupowane na 5 kategorii pod względem rozmiaru oraz – niezależnie od tego podziału – na 4 kategorie opisujące dziedziny zastosowania: Generatory diagramów i wizualizacja danych (DGDV), Wytwarzanie oprogramowania (DEV), Oprogramowanie aplikacyjne (APP) oraz

Systemy client-server (CSS). W celu dokonania oceny jakości tych systemów obliczono wartości 20 metryk, dotyczących pięciu kryteriów: rozmiaru, złożoności, zależności, abstrakcji danych, spójności i dziedziczenia. Ponadto systemy te poddano analizie pod kątem występowania 15 przykrych zapachów przy pomocy trzech narzędzi stosujących różne mechanizmy detekcji.

W wyniku tych badań stwierdzono, że rozkład przykrych zapachów jest podobny we wszystkich analizowanych dziedzinach zastosowań. Obserwacja ta oparta jest jednak tylko na detekcji za pomocą narzędzi i nie uwzględnia przypadków fałszywie pozytywnych. Ręczna walidacja wyników wykazała dużą liczbę takich przypadków w niektórych dziedzinach. Pokrywa się to z przypadkami zidentyfikowanymi w innych pracach, np. dotyczących zapachów *God Class* i *Data Class*. Wskazuje to na potrzebę uwzględnienia w procesie wykrywania przykrych zapachów czynników kontekstowych, np. dziedziny zastosowań, a także zidentyfikowania dziedzinowych przykrych zapachów.

Ponadto, w pracy tej zbadano związek obecności przykrych zapachami z wartościami niektórych metryk. W tym celu zidentyfikowano 5 systemów zawierających najwięcej i 5 systemów zawierających najmniej przykrych zapachów. Na tej podstawie określono metryki silnie skorelowane z obecnością dużej liczby zapachów: FANOUT, ATFD oraz WMC. Analiza poszerzona o poszczególne dziedziny zastosowań wykazała, że oprogramowanie aplikacyjne (APP) posiada największą wariancję metryk złożoności i zależności spośród wszystkich systemów.

Z kolei analiza korelacji pomiędzy częstością występowania przykrych zapachów i niektórych metryk także wskazała na istnienie pomiędzy nimi związków. Część z nich można wyjaśnić użyciem poszczególnych metryk w procesie wykrywania danego zapachu, jednak dla niektórych par zapachów związek ten nie jest oczywisty i stanowi kolejne potwierdzenie, że niektóre przykre zapachy są związane z wybranymi własnościami jakościowymi, które można ocenić za pomocą metryk.

Aby ocenić związek wielu metryk na obecność przykrego zapachu zastosowano także analizę głównych składowych (PCA). Wyniki w części pokrywały się z rezultatami analizy korelacji, natomiast w niektórych przypadkach wskazały na istnienie związków i niejawnych interakcji pomiędzy różnymi właściwościami badanego kodu, które można odkryć uwzględniając ich grupy, a nie pojedyncze metryki.

W pracy tej wskazano, że identyfikacja przykrych zapachów wymaga zastosowania nie tylko analizy metryk jako podstawowego źródła informacji, ale także uwzględnienia czynników kontekstowych, np. dziedziny zastosowania analizowanego systemu lub jego rozmiaru. Ich znajomość może pozwolić na poprawę dokładności procesu identyfikacji przykrych zapachów oraz ograniczenie liczby przypadków fałszywie pozytywnych. Dlatego ważnym kierunkiem badań jest identyfikacja przyczyn takiego zachowania, np. w celu ich opracowania narzędzi do ich szybkiego identyfikowania.

Dlatego w pracy [A5] zaproponowano taksonomię takich przykrych zapachów, które, przy uwzględnieniu uwarunkowań kontekstowych, mogą nie być związane z problemami pielęgnacyjnymi, czyli przykładów fałszywie pozytywnych. Taksonomia ta obejmuje dwie główne kategorie, które następnie dzielą się nad podkategorie:

1. Przykre zapachy wprowadzone świadomie jako nie dający się pominąć efekt uboczny rozwoju systemu:
   1.1. Wprowadzone w wyniku zastosowania wzorca projektowego;
   1.2. Wprowadzone przez język oprogramowania;
   1.3. Wprowadzone przez użycie bibliotek i ram aplikacyjnych;
   1.4. Będące wynikiem optymalizacji kodu;
   1.5. Powstałe w wyniku przeniesienia i adaptacji kodu z innego języka (np. ze strukturalnego do obiektowego);
   1.6. Odziedziczone po kodzie istniejącym i zastanym.
2. Przykre zapachy wprowadzone w sposób niezamierzony:
   2.1. Wprowadzone w wyniku generacji kodu, np. z użyciem generatorów na podstawie modeli lub szablonów;
   2.2. Wynikające ze sposobu reprezentacji programu, np. w postaci źródłowej lub pośredniej;
   2.3. Wynikające z niewłaściwie zdefiniowanego zakresu analizy, np. przeanalizowania testów jednostkowych przez narzędzia służące do analizy kodu produkcyjnego;

Konkretne wystąpienia przykrego zapachu należące do poszczególnych podkategorii mogą inaczej wpływać na pielęgnowalność, dlatego nie należy ich traktować w ten sam sposób jak pozostałych instancji zapachów. Na przykład, tworzenie klas pełniących rolę jedynie kontenerów danych (czyli spełniających kryteria obecności przykrego zapachu *Data Class*) jest rekomendowaną praktyką i wzorcem *Data Transfer Object* stosowanym podczas tworzenia rozproszonych aplikacji wielowarstwowych. Jest to przykład czynnika kontekstowego, który powinien być uwzględniony w procesie wykrywania tego przykrego zapachu

W pracy [A5], a także raporcie [B12], który rozszerza i uszczegóławia materię omawianą w tej pracy, przeanalizowano 12 przykrych zapachów oraz wskazano przykłady fałszywie pozytywnych przypadków ich występowania, klasyfikując je zgodnie z zaproponowaną taksonomią. Przykłady te mogą posłużyć do skonstruowania filtrów, które pozwolą poprawić dokładność wykrywania przykrych zapachów przez istniejące detektory. Obecnie trwają badania dotyczące empirycznej oceny takich filtrów do wykrywania przykrych zapachów prawdziwie i fałszywie pozytywnych.

3 Pozostałe obszary badań I zainteresowań naukowych

Poza głównym kierunkiem badań, dotyczącym analizy przykrych zapachów w kodzie programów, prowadziłem także badania w innych obszarach inżynierii

oprogramowania. Poniżej znajduje się lista publikacji dotyczących tych obszarów, a w kolejnych sekcjach – krótki przegląd dwóch najważniejszych z nich.

3.1 Lista pozostałych prac opublikowanych po uzyskaniu stopnia doktora

[B1] Błażej Pietrzak, **Bartosz Walter**, 2005, *Exploring Bad Code Smells Dependencies*, Proc. of Software Engineering: Evolution and Emerging Technologies Conference, 2005, IOS Press, pp. 353-364 (**MNiSW: 10 pkt**).

[B2] Jerzy R. Nawrocki, Łukasz Olek, Michal Jasiński, Bartosz Paliswiat, **Bartosz Walter**, Błażej Pietrzak, Piotr Godek, 2005, *Balancing Agility and Discipline with XPrince*, Proc. of RISE Conference, 2005, Springer LNCS, pp. 266-277 (**MNiSW: 13 pkt**).

[B3] Bartosz Bogacki, **Bartosz Walter**, 2006, *Evaluation of Test Code Quality with Aspect-Oriented Mutations*, Proc. of XP 2006 Conference, 2006, Springer LNCS, pp. 202-204 (**MNiSW: 13 pkt**).

[B4] Marcin Wolski, **Bartosz Walter**, Szymon Kupinski, Patryk Prominski, 2016, *One Metric to Combine Them All: Experimental Comparison of Metric Aggregation Approaches in Software Quality Models*, Proc. of IWSM-Mensura Conference, 2016, pp.159-163 (**MNiSW: 15 pkt**).

[B5] Szymon Kupinski, **Bartosz Walter**, Marcin Wolski, Jakub Chojnacki, 2017, *Filling the gaps: imputation of missing metrics' values in a software quality model*, Proc. of IWSM-Mensura Conference, 2017, pp. 82-87 (**MNiSW: 15 pkt**).

[B6] Marcin Wolski, **Bartosz Walter**, Szymon Kupiński, Jakub Chojnacki, *Software quality model for a research-driven organization - An experience report*, Journal of Software: Evolution and Process, Vol. 30 (2018) No. 5, pp. e1911 (**MNiSW: 20 pkt**).

[B7] Zarko Stanisavljevic, **Bartosz Walter**, Maja Vukasovic, Andrijana Todosijevic, Maciej Łabędzki, Marcik Wolski, GÉANT Software Maturity Model. Proc. of TELFOR 2018 Conference, 2018

[B8] Francesca Arcelli Fontana, Marco Zanoni, **Bartosz Walter**, Paweł Martenka: Code Smells, Micro Patterns and their Relations. ERCIM News 2012 (88).

[B9] **Bartosz Walter**, Paweł Martenka: Looking for Patterns in Code Bad Smells Relations. Proc. of ICST Workshops, 2011, pp. 465-466.

[B10] Paweł Martenka, **Bartosz Walter**, Hierarchical Model for Evaluating Software Design Quality. E-Informatica 4(1), 2010, pp. 21-30.

[B11] **Bartosz Walter**, Błażej Matuszyk, Francesca Arcelli Fontana, *Including structural factors into the metrics-based code smells detection*. Proc. of XP Workshops 2015, p. 11.

[B12] Francesca Arcelli Fontana, Jens Dietrich, **Bartosz Walter**, Aiko Yamashita, M. Zanoni, *Preliminary catalogue of anti-pattern and code smell false positives. Technical Report RAT-5/15, 2015.* http://www2.cs.put.poznan.pl/wp-content/uploads/2015/11/RA-5-2015.pdf

3.2 Zastosowanie programowania aspektowego w testowaniu mutacyjnym

Testowanie mutacyjne jest techniką pozwalającą na ocenę jakości zestawu testów i jego efektywności w wykrywaniu defektów oprogramowania. Opiera się ona na założeniu, że dowolna modyfikacja kodu systemu (zwana mutantem) powinna spowodować negatywny wynik wykonania przynajmniej jednego przypadku testowego, o ile zbiór przypadków testowych dla tego systemu jest spójny i kompletny. Jeżeli natomiast mutant przetrwa wszystkie testy, wówczas oznacza to, że zbiór przypadków testowych jest niepełny.

Zastosowanie testowania mutacyjnego wymaga jednak złożonego aparatu generującego mutanty oraz wykonującego testy, co negatywnie rzutuje na czas wymagany do wygenerowania mutacji i ich weryfikacji. W rezultacie prowadzi to do ograniczonego zastosowania tej techniki w praktyce.

Zaproponowane w pracy [B3] podejście wykorzystuje w tym celu programowanie aspektowe (Aspect-Oriented Programming, AOP [Kiczales et al., 2001]), które pozwala na precyzyjne łączenie ze sobą fragmentów kodu dotyczących różnych zagadnień. Zaimplementowanie metod generowania mutacji (czyli operatorów mutacyjnych) w postaci aspektów, które następnie będą zmieniać zachowanie programu w sposób deterministyczny lub niedeterministyczny, pozwala na znaczne ograniczenie czasu wykonywania testów mutacyjnych oraz umożliwia rozszerzenie ich o nowe operatory. Wyniki badań wskazują, że programowanie aspektowe jest efektywną metodą pozwalającą na praktyczną realizację testowania mutacyjnego.

3.3 Modele jakości oprogramowania i dojrzałości zespołów wytwarzających oprogramowanie

We współpracy z Poznańskim Centrum Superkomputerowo-Sieciowym w latach 2015-2018 opracowano hierarchiczny model jakości oprogramowania, oparty na klasycznych modelach Boehma (1978) i McCalla (1977), jednocześnie uwzględniający specyficzne cechy organizacji, w której to oprogramowanie powstaje. GEANT jest projektem, w ramach którego powstała platforma współpracy operatorów europejskich sieci infrastrukturalnych służących do zastosowań naukowych i edukacyjnych. Produkty informatyczne, które powstają w ramach tej organizacji, dostarczają zaawansowanych, prototypowych usług opartych na możliwościach wysokowydajnej sieci informatycznej. Rozwój tego oprogramowania odbywa się w zespołach złożonych z inżynierów i naukowców z różnych krajów, pracujących w oddaleniu od siebie i komunikujących się jedynie za pomocą sieci, co znacząco wpływa na efekt końcowy,

Istniejące modele jakości oprogramowania nie uwzględniają wszystkich cech specyficznych dla organizacji GEANT. W związku z tym konieczne było stworzenie takiego modelu jakości, który całościowo ujmowałby charakterystyki jakościowe szczególnie istotne w tego typu organizacji, pozwalając na ocenę powstających w ten sposób produktów.

W ramach prowadzonych prac badawczych zidentyfikowano charakterystyki, podcharakterystyki oraz metryki mierzące wielkości istotne dla różnych wymiarów jakościowych oraz określono relacje występujące pomiędzy nimi. Rozwiązując praktyczne problemy napotkane podczas przygotowywania modelu jakości do wdrożenia, podjęto badania w niektórych pobocznych obszarach. Model jakości zaproponowany dla GEANT ma strukturę hierarchiczną, wewnątrz której zachodzi agregacja zmierzonych wartości na poszczególnych poziomach. W celu wyboru sposobu agregacji, w pracy [B4] dokonano przeglądu i porównania wybranych metod opartych na miarach zróżnicowania, i wskazano te, które najbardziej odpowiadają potrzebom modelu.

Drugim istotnym problemem, często spotykanym w zastosowaniach praktycznych, jest częściowa niedostępność danych do zagregowania i konieczność ich oszacowania (imputacji) na podstawie istniejących informacji. Ma to istotny wpływ na wynik agregacji. Istnieje wiele metod imputacji, charakteryzujących się różną odpornością na zakłócenia oraz na niedokładność danych użytych do oszacowania. W pracy [B5] przedstawiono przegląd i eksperymentalną ocenę przydatności kilku popularnych metod imputacji, z uwzględnieniem potrzeb i ograniczeń występujących w GEANT.

Całościowe wyniki prac dotyczących modelu jakości oprogramowania zostały zaprezentowane w [B6] i są obecnie są wdrażane w ramach organizacji GEANT. Równolegle do tego modelu jakości opracowano także model dojrzałości zespołów (Software Maturity Model for GEANT), w których oprogramowanie jest wytwarzane [B7]. Ma on za zadanie zidentyfikować te elementy w ramach cyklu rozwojowego oprogramowania, które są kluczowe dla skutecznej realizacji przez zespół zadań obarczonych wysokim ryzykiem niepowodzenia, przy ograniczeniach wynikających ze specyfiki organizacji. Model dojrzałości wskazuje cele, jakie zespoły powinny realizować, jednocześnie pozostawiając im swobodę w doborze metod osiągnięcia tych celów. Model, po jego wdrożeniu, pozwoli nie tylko na identyfikację słabych i mocnych stron poszczególnych zespołów, lecz także na poprawę koordynacji metod wytwarzania oprogramowania stosowanych przez poszczególne zespoły oraz ułatwi identyfikację I wymianę dobrych praktyk.

4 Działalność dydaktyczna, organizacyjna i popularyzatorska

Pracując na Wydziale Informatyki (a wcześniej na Wydziale Elektrycznym oraz Wydziale Informatyki i Zarządzania) Politechniki Poznańskiej prowadziłem zajęcia wykładowe i laboratoryjne dla studentów I i II stopnia z przedmiotów dotyczących wybranych obszarów inżynierii oprogramowania: projektowania i modelowania oprogramowania, doskonalenia procesów programistycznych, zarządzania jakością i aplikacji internetowych. Prowadzone przeze mnie zajęcia były wysoko oceniane przez studentów (średnia: 4.5 – 4.8, w skali 1 min – 5 max). W 2014 roku otrzymałem nagrodę Rektora Politechniki za osiągnięcia dydaktyczne.

Od czasu uzyskania stopnia doktora wypromowałem około 65 magistrów. Pełniłem również rolę opiekuna naukowego dla 2 osób, a obecnie pełnię funkcję promotora pomocniczego w przewodzie doktorskim mgr. inż. Tareka Alkhaeira (Wydział Informatyki Politechniki Poznańskiej, promotor: dr hab. inż. Andrzej Jaszkiewicz, prof. nadzw.), wszczętym w kwietniu 2019 roku.

W 2011 roku z mojej inicjatywy na Wydziale Informatyki zostało powołane Studium Podyplomowe Inżynierii Oprogramowania, które następnie zorganizowałem, oraz w którym od początku pełnię rolę kierownika. W ciągu ośmiu edycji, jakie miały miejsce do 2019 roku, studium ukończyło ok. 250 słuchaczy.

Poza obowiązkami naukowymi i dydaktycznymi wykonywałem również prace organizacyjne. W latach 2009-2011 byłem koordynatorem i kierownikiem funkcjonującego na Politechnice Poznańskiej Centrum Wsparcia w zakresie technologii Eclipse, prowadzonego w partnerstwie i na zlecenie IBM Polska sp. z o.o.. Centrum świadczy klientom z całego świata usługi w zakresie wsparcia trzeciej linii (L3) dla produktów IBM opartych na platformach Eclipse i Jazz, a także wykonuje prace badawczo-rozwojowe. Centrum zatrudnia na stałe kilkoro wysoko wykwalifikowanych inżynierów oprogramowania.

W latach 2010-2015 byłem koordynatorem projektu dotyczącego testowania oprogramowania, realizowanego przez Wydział Informatyki na zlecenie Roche Polska sp. z o.o. W projekcie uczestniczyło przez ten czas ok. 40 osób, projektując, planując, wykonując i raportując testy oprogramowania w ściśle regulowanym środowisku farmaceutycznym.

Ponadto, uczestniczyłem w organizacji kilku konferencji i warsztatów jako przewodniczący komitetu lub współorganizator (CEE-SET 2007 w Poznaniu, MaLTeSQuE 2017, 2018 i 2019). Byłem także członkiem komitetów programowych kilkunastu konferencji i warsztatów. Współredagowałem materiały konferencji CEE-SET 2007 i 2008 (wydane przez Springer Verlag w serii LNCS).

Dodatkowym elementem mojej działalności jest popularyzacja zagadnień naukowych. W latach 2015, 2016 i 2017 brałem udział w międzynarodowym konkursie popularyzacji nauki FameLab, osiągając w 2017 roku poziom finału krajowego. Ponadto, jestem autorem kilku artykułów popularyzatorskich; przeprowadziłem także kilka prezentacji i warsztatów popularyzatorskich.