Załącznik 2 A

Warszawa 11 marca 2015

Dr inż. Ilona Ewa Bluemke
Instytut Informatyki
Wydział Elektroniki i Technik Informacyjnych
Politechnika Warszawska

AUTOREFERAT

1. Posiadane stopnie naukowe – z podaniem nazwy, miejsca i roku ich uzyskania

1989 - doktor nauk technicznych – dyscyplina informatyka, Politechnika Warszawska, praca: „Lokalne strategie rekonfiguracji i ich wpływ na niezawodność i wydajność macierzy elementów przetwarzających", promotor prof. dr hab. Krzysztof Sapiecha.

1978 - magister inżynier, specjalność budowa i oprogramowanie maszyn cyfrowych, Instytut Informatyki, Wydział Elektroniki Politechniki Warszawskiej, praca: „Preprocesor formuł na język PLAN dla maszyn ODRA serii 1300", promotor prof. dr hab. Włodzimierz Zuberek.

2. Zatrudnienie

1990 -	adiunkt, Instytut Informatyki, Politechniki Warszawskiej
1980 - 1990	starszy asystent Instytut Informatyki PW
1978 - 1980	programista, Centralny Ośrodek Obliczeniowy Politechniki Warszawskiej

3. Wskazanie osiągnięcia wynikającego z art. 16. Ust. 2 ustawy z dnia 14.03.2003 o stopniach naukowych i tytule naukowym oraz o stopniach i tytule w zakresie sztuki (Dz.U. nr 65, poz. 595 ze zm.,)

Od 2001 roku prowadziłam prace badawcze dotyczące szeroko pojmowanej **inżynierii oprogramowania** ze szczególnym uwzględnieniem **problematyki poprawiania jakości oprogramowania.**

Przedstawionym **osiągnięciem naukowym jest opracowanie i dogłębna analiza metod pozwalających na kompleksowe i wielokryterialne poprawianie jakości oprogramowania oraz zaprojektowanie i zrealizowanie narzędzi umożliwiających praktyczne zastosowanie tych metod.**

3.1 Tytuł osiągnięcia naukowego:

Jakość oprogramowania w ujęciu wielokryterialnym

3.2 Osiągnięcie naukowe – złożone z cyklu publikacji naukowych:

Wykaz publikacji stanowiących osiągnięcie naukowe

[H1] **I. Bluemke**: "Object oriented metrics useful in the prediction of class testing complexity", *Proc. of 27th IEEE Euromicro Conf.*, Workshop on Component Based Software Engineering, Warszawa 4-6 września 2001, str. 130-136 , DOI: 10.1109/EURMIC.2001.952447

[H2] **I. Bluemke**, K. Billewicz: "Aspects in the maintenance of compiled programs", *Proc. of Int. Conf. on Dependability of Computer Systems DepCoS-RELCOMEX 2008*, Szklarska Poręba, 25-29 czerwiec 2008, str. 253-260, DOI: 10.1109/DepCoS-RELCOMEX.2008.15

[H3] **I. Bluemke**, K. Billewicz: "Aspect modification of an EAR application", *Advanced Techniques in Computing Sciences and Software Engineering*, ed. K. Elleithy, Springer 2010, str.105-110, DOI:10.1007/978-90-481-3660-5_18

[H4] **I. Bluemke**, A. Orlewicz: "Knowledge mining with ELM system", *Knowledge-Based and Intelligent Information and Engineering Systems*, R. Setchi et al. (eds.), LNAI vol. 6277, part II, Springer 2010, str. 369-378, DOI:10.1007/978-3-642-15390-7_9

[H5] **I. Bluemke**, M. Tarka: "Detection of anomalies in SOA system by learning algorithms", *Complex Systems and Dependability*, W. Zamojski et al. (eds.), Advances in Intelligent and Soft Computing, vol. 170, Springer 2012, str. 69-85, DOI: 10.1007/978-3-642-30662-4_5

[H6] **I. Bluemke**, M. Tarka: "Learning Algorithms in the Detection of Unused Functionalities in SOA Systems", *Computer Information Systems and Industrial Management*, S. Khalid et al. (eds.), LNCS vol. 8104, Springer 2013, str. 389-400, DOI:10.1007/978-3-642-40925-7_36

[H7] **I. Bluemke**, M. Tarka:"Detection of changes in group of services in SOA system by learning algorithms", *New Research in Multimedia and Internet Systems*, A. Zgrzywa, K. Choroś, A. Siemiński (red.), part IV, *Advances in Intelligent Systems and Computing*, vol. 314, Springer 2014, str. 215-225, DOI:10.1007/978-3-319-10383-9_20

[H8] **I. Bluemke**, A. Rembiszewski: "Dataflow approach to testing Java programs", *Proc. of Int. Conf. on Dependability of Computer Systems DepCoS - RELCOMEX*, Brunów, Poland, 30 June – 2 July 2009, str. 69-76, DOI:10.1109/DepCoS-RELCOMEX.2009.34

[H9] **I. Bluem**ke, K. Kulesza: "A Comparison of Dataflow and Mutation Testing of Java Methods", "*Dependable Computer System*", W. Zamojski et al. (eds.), *Advances in Intelligent and Soft Computing*, vol. 97, Springer 2011, str. 17-30, DOI: 10.1007/978-3-642-21393-9_2

[H10] **I. Bluemke**, K. Kulesza: "Reduction in mutation testing of Java classes", *Proc. of ICSOFT-EA 2014 – Ninth Int. Conf. on Software Engineering and Applications*, A. Holzinger, T. Libourel, L. Maciaszek, S. Mellor (eds.), SCITEPRESS, str. 297-304, DOI: 10.5220/0004992102970304

3.3 Ocena wkładu pracy w publikacje należące do osiągniecia naukowego

Wszyscy współautorzy prac, należących do osiągniecia naukowego **byli studentami**, którzy pod moim kierunkiem wykonywali prace dyplomowe (magisterskie i inżynierskie). Udział studentów sprowadzał się do wykonania eksperymentu (wg mojego projektu) i implementacji oprogramowania według mojego projektu.
[H1] - autor 100%, a prace: [H2-H10] - pierwszy autor , wkład 60%- 80%.

3.4. Opis osiągnięcia naukowego

Przedmiotem niniejszej pracy są metody pozwalające na **kompleksowe i wielokryterialne poprawianie jakości oprogramowania**. Badania zostały przeprowadzone w Instytucie Informatyki Politechniki Warszawskiej w latach 2001-2014.

Stale zwiększająca się złożoność oprogramowania powoduje, że jakość oprogramowania staje się krytycznym jego aspektem. Ponadto, jest coraz trudniej zapewnić jakość oprogramowania na zadowalającym poziomie. Złożone systemy oprogramowania są przydatne tylko wtedy gdy są niezawodne, użyteczne, wydajne, funkcjonalne, adaptowalne, łatwe do utrzymania. Zapewnienie wysokiej jakości oprogramowania jest zagadnieniem niezwykle złożonym, zależnym od wielu aspektów, które było i jest przedmiotem wielu prac prowadzonych na całym świecie np. [Sur14, PoS14, RLR14, Sin13, WBB13, KGK12, RPB12, Bak11, Loc11, Glin09, MBR09, Eck08, Wag08, GSc07, Dei07, Kho07, Cot06, Sur03, Voa03, Pfl01, Kit96, Dro95, Gar87, Boe78, Bro78] oraz w Polsce np. [MOT14, KoN14, BTD14, GoŁ13, GJD12, Sac10, Msz07, Bil07, Kob05, MaK04, Kob03, Beg03, Dym00, Beg99, BeJ90, Hoff1, Hoff2].

Wiadomo, że ogromny wpływ na jakość oprogramowania ma inżynieria wymagań (ang. requirements engineering) np. [Ter13], [Glin09], [Kot00], [Bro78]. Jest bardzo wiele prac poświęconych właśnie tej tematyce np. [Ral13], [GJD12], [Eck08], [Nus00] a szeroką bibliografię dotyczącą inżynierii wymagań prowadził do 2011 roku A. M. Davis [Dav11]. Jednakże inżynieria wymagań nie jest zagadnieniem omawianym w niniejszej pracy.

Ocena jakości oprogramowania jest bardzo istotna zarówno w procesie wytwarzania, użytkowania a także zakupu oprogramowania. Składowymi oceny jakości oprogramowania są: modele jakości, metoda oceny oraz ocena (pomiar) oprogramowania.

Przedstawiony w niniejszym opracowaniu cykl publikacji z lat 2001- 2014 składa się z 10 artykułów, stanowiących istotny wkład w rozwój metod poprawiania jakości oprogramowania. Prace te dotyczą wielu aspektów poprawy jakości różnego typu oprogramowania, zarówno w fazie wytwarzania jak i użytkowania, pozwalają także na ocenę oprogramowania na podstawie obliczonych metryk. Praca [H1] dotyczy wyznaczania metryk obiektowych oprogramowania projektowanego w UML [UML] w narzędziu typu CASE (ang. Computer Aided Software Engineering) i na podstawie wartości metryk na lokalizacje miejsc w projekcie, których testowanie wymaga szczególnej uwagi i jest bardziej pracochłonne. Następne dwie prace ([H2, H3]) dotyczą modyfikacji istniejącego oprogramowania, nie wymagającej dostępu do jego kodu źródłowego i dostosowania go do potrzeb użytkownika. Kolejna praca ([H4]) dotyczy badania zachowania użytkownika korzystającego z oprogramowania i wnioskowania o jego potrzebach. Następne trzy prace ([H5, H6, H7]) dotyczą wykrywania różnego typu anomalii w systemach o organizacji serwisowej (ang. Service Oriented Architecture – SOA) [SOA1, SOA2]. Wreszcie prace [H8, H9, H10] dotyczą wybranych typów testowania oprogramowania, poprawy jego efektywności oraz zmniejszenia pracochłonności z nim związanego.

W następnej sekcji zamieszczono krótkie wprowadzenie w problematykę jakości oprogramowania a następnie każdy z artykułów [H1]- [H10] jest scharakteryzowany z zaznaczeniem kryteriów jakości oprogramowania na które wpływa.

3.4.1 Problem jakości oprogramowania

Jakość oprogramowania (ang. software quality) jest to ogół cech produktu programowego, które wpływają na zdolności spełniania przez niego określonych wymagań, takich jak np. elastyczność, funkcjonalność, integralność, niezawodność, efektywność, użyteczność czy wydajność.

Oprogramowanie jest produktem i jako taki powinno zaspokajać aktualne i przewidywalne potrzeby jego użytkowników.

Zagadnienia jakości oprogramowania są od wielu lat jednym z ważniejszych celów badań w informatyce co wynika z rosnących wymagań użytkowników i osób obsługujących systemy informatyczne. Prowadzono bardzo wiele prac dotyczących tematyki jakości oprogramowania np. [Sur14, PoS14, RLR14, Sin13, WBB13, KGK12, RPB12, Bak11, Loc11, Glin09, MBR09, Eck08, Wag08, GSc07, Dei07, Kho07, Cot06, MOT14, KoN14, BTD14, GoŁ13, GJD12, Beg99, Beg03, Bil 07, Cot06, Dym00, Gar87, Hoff1, Hoff2, Kan06, Kob03, Kob05, Voa03]. Szeroka dyskusja prowadzona w środowiskach producentów i użytkowników oprogramowania doprowadziła do powstania i przyjęcia norm międzynarodowych. Zaproponowano wiele modeli jakości oprogramowania zawierających zbiory atrybutów pozwalających na jego ocenę. Pierwsze takie modele powstały w końcu lat siedemdziesiątych, a ostatni w roku 2005. Modele jakości podlegają ewolucji związanej ze zmianami sposobu wytwarzania i charakteru oprogramowania.

Model McCall [McC77] z 1977 roku był pierwszym modelem jakości oprogramowania i wprowadził zestaw oczekiwanych charakterystyk jakości wraz z zestawem atrybutów/metryk, które na ową jakość mają wpływ. Model ten bazuje na cechach technicznych produktu. Model Mc Call'a dzieli kryteria oceny oprogramowania na grupy związane: ze sposobem działania oprogramowania, z możliwością wprowadzenia zmian i poprawek w programie oraz z mobilnością oprogramowania: Już wówczas zauważono, że problemem jest osiągnięcie optimum, gdyż niektóre z wyżej wymienionych cech są sprzeczne np. wzrost niezawodności oprogramowania często powoduje spadek efektywności.

Model zaproponowany w 1978 roku przez Boehm'a [Boe78] zmienił punkt widzenia jakości oprogramowania, przenosząc punkt ciężkości na perspektywę użyteczności oprogramowania. Według Pfleeger'a [Pfl01] jest to pierwsze wskazanie, że jakość oprogramowania może być postrzegana tylko wtedy, kiedy oprogramowanie jest użyteczne.

Kolejne modele, mniej znane, proponowali Dromey [Dro95] oraz Kitchenham i Pfleeger [Kit96].

3.4.1.1 Normy jakości oprogramowania

Normy jakości oprogramowania powstawały i były zatwierdzane począwszy od początku lat dziewięćdziesiątych. Pierwsza norma to standard ISO 9126 z 1991 [ISO91] definiujący charakterystyki jakości. Model opisany w normie określał sześć głównych charakterystyk jakości: 1. funkcjonalność (ang. functionality), 2. niezawodność (ang. reliability), 3. użyteczność (ang. usability), 4. wydajność (ang. efficiency), 5. łatwość utrzymania (ang. maintainability) i 6. przenośność (ang. portability).

Wyżej wymienione atrybuty występowały także we wcześniejszym modelu McCall'a. Norma ISO 9126 nie spełniła jednak pokładanych w niej nadziei [Pfl01], głównie ze względu na ograniczenie się do perspektywy producenta, brak stanowiska w sprawie całościowej oceny jakości oraz brak dokładnych wytycznych dotyczących mierzenia poszczególnych parametrów jakości.

Określenie jakości oprogramowania zostało zmienione w normie ISO/IEC 9126 z 2001 roku [ISO01] W normie wyróżniono trzy poziomy jakości i zdefiniowano trzy zbiory charakterystyk:

1. jakość wewnętrzna/statyczna (ang. internal quality),
2. jakość zewnętrzna/dynamiczna (ang. external quality),
3. jakość użytkowa (ang. in use quality).

Model zdefiniowany w tej normie uwzględnia również różne aspekty jakości dyskutowane we wcześniejszych modelach, czyli jakość produktu, jakość techniczną produktu oraz jakość procesu. Jakość zewnętrzna oprogramowania jest to jego charakterystyka postrzegana z

zewnątrz, wtedy gdy oprogramowanie zostało już ukończone. Jakość wewnętrzna jest to charakterystyka oprogramowania postrzegana od strony producenta. Jest ona bazą do szacowania przyszłej jakości zewnętrznej gotowego produktu. Model jakości wewnętrznej korzysta z tych samych atrybutów co model jakości zewnętrznej jednak dotyczą one produktów pośrednich uzyskanych w procesie wytwarzania oprogramowania.

Wewnętrzna i zewnętrzna jakość definiowana jest jako suma następujących charakterystyk (analogicznych do poprzedniego wydania normy 9126): funkcjonalność (ang. functionality), niezawodność (ang. reliability), użyteczność (ang. usability), wydajność (ang. efficiency), utrzymywalność (ang. maintainability) i przenośność (ang. portability).

Jakość użytkowa jest to charakterystyka oprogramowania postrzegana od strony użytkownika produktu/klienta. Składa się na nią zbiór cech pozwalających na wydajne, efektywne, bezpieczne i satysfakcjonujące korzystanie z produktu. Definiowana jest jako suma następujących charakterystyk: przydatność (ang. effectiveness), produktywność (ang. productivity), bezpieczeństwo (ang. safety) i zdolność do zaspokojenia potrzeb (ang. satisfaction).

Na podstawie normy ISO/IEC serii 9126 powstał model jakości SQuaRE (Software product Quality Requirements and Evaluation) ISO 25000:2005 [ISO05]. Ta norma nie tylko porządkuje model oceny jakości oprogramowania opisany w ISO/IEC 9126:2001 [ISO01], ale w sposób istotny go rozszerza. Na najwyższym poziomie modelu jakości oprogramowania pozostawiono podział na jakość wewnętrzną, zewnętrzną i użyteczną. **Jakość wewnętrzna i zewnętrzna** oprogramowania zdefiniowana jest analogicznie do definicji w ISO/IEC 9126:2001, z tym że zmieniono charakterystyki wysokiego poziomu – zamiast sześciu w nowym modelu jest osiem atrybutów: 1. funkcjonalność (ang. functionality), 2. poufność (ang. security), 3. zgodność techniczna (ang. interoperability), 4. niezawodność (ang. reliability), 5. użyteczność (ang. usability), 6. wydajność (ang. efficiency), 7. łatwość utrzymania (ang. maintainability) i 8. przenośność (ang. portability).

Oprogramowanie powinno jak najlepiej zaspokajać aktualne i przewidywalne potrzeby jego użytkowników toteż bardzo ważna jest wysoka jakość użytkowa (ang. in use quality). **Jakość użyteczna** oprogramowania, została zdefiniowana na nowo w modelu SQuaRE. Jej główne charakterystyki to: użyteczność (ang. usability in use), zgodność z kontekstem (ang. context in use), bezpieczeństwo (ang. safety in use), poufność (ang. security in use) oraz łatwość użycia (ang. adaptability in use).

3.4.1.2 Modele jakości oprogramowania w literaturze

Oprócz modeli jakości będących normą, w literaturze można znaleźć wiele modeli jakości oprogramowania ogólnego lub określonego typu, czy też modeli jakości poszczególnych czynności procesu produkcji oprogramowania. Lochmann i Goeb [Loc11] zaproponowali meta model jakości pozwalający na łączenie różnych ogólnych idei jakości znajdujących się w normach oraz modeli jakości dziedzinowych. Model ten pozwala na wyrażanie relacji między różnymi dziedzinami inżynierii oprogramowania np. inżynieria wymagań i testowanie na jakość oprogramowania. Deissenboeck i inni [Dei07] zaproponowali w 2007 model jakości dla utrzymywalności (ang. maintainability) oparty o wybrane czynności (ang. activity-based quality model - ABQM for maintainability) wykonywane w procesie utrzymywania oprogramowania. Modele oparte o czynności zostały także opracowane dla bezpieczeństwa [Wag09], użyteczności [Win08] oraz wymagań jakościowych oprogramowania [Wag08] a także dla specjalnych typów programów np. oprogramowania „webowego” [Wag09]. W pracy [Kho07] zaproponowano model klasyfikacji jakości oprogramowania oparty o programowanie genetyczne, w pracy [Bak11] probabilistyczny model jakości a w pracy [Zha08] model dwuwymiarowy. Modele te są interesujące ale brak informacji o ich

zastosowaniu w praktyce. Porównanie kilku podstawowych, wymienionych w sekcji 3.4.1 modeli można znaleźć w [Sin13].

3.4.2 Jakość oprogramowania w ujęciu wielokryterialnym

Każdy z modeli jakości oprogramowania proponuje pewien zbiór kryteriów (atrybutów), jedne są mierzalne np. niezawodność, inne są oceniane subiektywnie. W różnych typach oprogramowania kryteria te mogą mieć inne wagi np. kryterium łatwego utrzymania dla pewnych systemów nie jest ważne, za to krytyczna jest niezawodność, w a przypadku innego typu oprogramowania łatwe utrzymanie może mieć wysoka wagę. Stąd, mimo istnienia wielu modeli, także norm międzynarodowych, w praktyce trudno jest zdefiniować jakość oprogramowania, obliczyć ją, przewidzieć, czy zapewnić mimo że istnieje bardzo wiele prac (cześć wymieniono w sekcji 3.4.1), książek na ten temat np. „Handbook of Software Quality Assurance" - [GSc07].

	praca	***H1***	***H2***	***H3***	***H4***	***H5***	***H6***	***H7***	***H8***	***H9***	***H10***
Jakość wewnętrzna i zewnętrzna	funkcjonalność		v	v	v		v				
	poufność					v		v			
	zgodność techniczna	v							v	v	v
	niezawodność	v				v	v	v	v	v	v
	użyteczność				v	v					
	wydajność					v					
	łatwość utrzymania		v	v					v	v	v
	przenośność		v								
	praca	*H1*	*H2*	*H3*	*H4*	*H5*	*H6*	*H7*	*H8*	*H9*	*H10*
Jakość użyteczna/	użyteczność		v	v	v						
	zgodność z kontekstem	v				v					
	bezpieczeństwo	v				v	v	v	v	v	v
	poufność					v	v	v	v	v	v
	łatwość użycia		v	v	v						

Tabela 1. Powiązanie prac z cyklu z kryteriami jakości normy ISO 25000:2005

W niniejszej pracy, przyjęto dla jakości oprogramowania **zbiór kryteriów z aktualnej normy** ISO 25000:2005. W Tabeli 1 pokazano związek między omawianymi pracami oraz kryteriami jakości aktualnej normy ISO 25000:2005. Omawiane prace dotyczą wszystkich kryteriów jakości. Zwiększanie jakości oprogramowania może odbywać się poprzez ulepszanie poszczególnych składników jego jakości wewnętrznej i zewnętrznej oraz użytecznej. **Prace tworzące cykl ([H1] –[H10]) pozwalają na kompleksowe poprawianie jakości zewnętrznej, wewnętrznej i użytkowej oprogramowania w zakresie wszystkich kryteriów** normy ISO.

W dalszej części każdy z artykułów wchodzący w skład cyklu jest scharakteryzowany z zaznaczeniem istotnego wkładu w dziedzinę poprawy jakości oprogramowania. Opis prac został podzielony na następujące części:

1. dotycząca oceny projektowanego oprogramowania,
2. wykorzystujące programowanie aspektowe do poprawiania kryteriów jakości,
3. wykorzystujące algorytmy sztucznej inteligencji do poprawiania kryteriów jakości,
4. dotyczące testowania.

3.4.2.1 Ocena oprogramowania w fazie projektowania [H1]

Ogromny wpływ na jakość oprogramowania ma jakość projektu. Istotne jest, by już we wczesnych fazach procesu projektowania, można było wykrywać i eliminować błędy, braki, gdyż im wcześniej wykonujemy zmianę, tym jej koszt wprowadzenia jest mniejszy. Pewne miary oprogramowania [Kan06] świadczą o złożoności oprogramowania a także pozwalają wskazać miejsca w projekcie o większym prawdopodobieństwie powstawania błędów, miejsca trudniej testowalne a więc miejsca w których trudniej będzie zachować zgodność techniczną czy zapewnić wysoką niezawodność. W pracy [H1], zaprezentowanej na 27 konferencji IEEE Euromicro (2001), pokazałam metryki obiektowe, niezależne od języka programowania, które można policzyć już we wstępnych krokach realizacji projektu a pozwalające na oszacowanie wysiłku związanego z testowaniem i wykrywanie miejsc w projekcie które mogą być bardziej podatne na powstawanie błędów (ang. error prone). W pracy tej zaproponowałam obliczanie metryk zintegrowane z narzędziem CASE. Pokazałam jak metryki Chidamber-Kemerer [ChK94], często stosowane dla oprogramowania obiektowego, można policzyć mając wczesne stadium projektu systemu w UML [UML]. Zrealizowałam rozszerzenia (ang. plug in) do istniejących w owym czasie narzędzi CASE pozwalające na obliczanie tych metryk (dla systemu Paradigm Plus [H1], później dla systemu Select Enterprice i Rational Rose [Blu04]). Ówczesne narzędzia nie posiadały takich możliwości. We współcześnie stosowanych narzędziach CASE np. Software Architect liczenie metryk jest już wbudowane.

W pracy [H1] zaproponowałam także metrykę obiektową - *Length of Message Sequence (LMS)* określającą długość sekwencji komunikatów w diagramach sekwencji. Sekwencje komunikatów modelowane na diagramach sekwencji są od dawna wykorzystywane w testowaniu oprogramowania ([JoE94], [KiT94]). Im dłuższe są te sekwencje, tym większa jest wartość metryki LMS, tym dłuższe będzie testowanie integracyjne obiektów modelowanego oprogramowania (dłuższy czas potrzebny na wymuszenie i wykonanie ścieżek komunikatów i metod). Problemem oceny złożoności testowania w oparciu o metryki obiektowe zajmowałam się także w pracy [Blu01], w której zaproponowałam inną metrykę obiektową dotyczącą klasy - STC – State Transition Complexity. Metryka STC (złożoność cyklomatyczna) określa złożoność maszyny stanowej modelującej zachowanie klasy. Testowanie klas w oparciu o ich model stanowy, zaproponowane przez Turner'a i Robson'a [TuR93], wymaga przygotowania przypadków testowych badających zgodność zachowania obiektów klasy z ich modelem (maszyną stanową). Większa liczba stanów i przejść będzie wymagała przygotowania większej liczby testów.

Stosowanie metryk obiektowych już we wczesnych krokach produkcji oprogramowania pozwala na zwiększanie niezawodności, zgodności technicznej (kryteria jakości zewnętrznej i wewnętrznej), oraz zgodności z kontekstem, bezpieczeństwa (jakość użyteczna) .

3.4.2.2 Wykorzystanie programowania aspektowego [H2,H3]

O tym, że modularyzacja jest podstawą prawidłowej pielęgnacji kodu, pisali najwięksi badacze inżynierii oprogramowania, m.in. D. Parnas [Par72] i E. Dijkstra [Dij76]. Postulowali oni dekompozycję programu na części, z których każda będzie dotyczyła jednego

zagadnienia. Kolejne paradygmaty programowania, począwszy od programowania funkcyjnego, poprzez strukturalne, aż po obiektowe, próbowały spełnić ten postulat. W kolejnych generacjach języków i metod programowania pojawiały się nowe koncepcje, które dzieliły program według różnych kryteriów. W programach obiektowych w jednej metodzie może znajdować się zarówno kod odpowiedzialny za logikę biznesową jak i za inne zagadnienia np. logowanie, bezpieczeństwo czy zapis danych do bazy. Tego typu organizacja kodu powoduje jego zagmatwanie, rozproszenie podobnych znaczeniowo elementów w różnych miejscach, trudności w śledzeniu jego wykonania czy też utrudnienia w dalszym rozwoju programu. Programowanie aspektowe ma pomóc w eliminacji tych problemów.

W programowaniu aspektowym program powinien być zdekomponowany na zagadnienia (ang. concern), czyli kwestie, funkcje, które muszą zostać rozwiązane, aby osiągnąć cel stawiany systemowi. Każde z zagadnień powinno być oprogramowane w odrębnym aspekcie (ang. separation of concern). Koncepcja programowania aspektowego początkowo rozwijana była w laboratorium Xerox PARC (Paolo Alto Research Center) w latach 1995 – 1996 a została opublikowana w 1997w pracy [Kic97]. Proces rozbicia wymagań na składowe zwane zagadnieniami przecinającymi nazywany jest aspektową dekompozycją (ang. aspectual decomposition). Składowe odpowiadają niezależnym logicznie aspektom programu. Poza logiką programu (zwaną logiką biznesową), może występować wiele pobocznych zagadnień, ortogonalnych względem siebie i logiki biznesowej np. bezpieczeństwo, logowanie. Program aspektowy składa się z kodu napisanego w języku obiektowym np. w Javie [Java] oraz wplecionego weń kodu aspektowego w języku aspektowym np. AspectJ [Lad03]. Programowanie aspektowe pozwala na zaaplikowanie nowego kodu do kodu już istniejącego w procesie wplatania (ang. weaving). Zdefiniować trzeba kod, który ma zostać zaaplikowany (rada, ang. advice) oraz określić miejsca jego wplecenia czyli punkty złączeń (ang. joinpoint). Programowanie aspektowe zastosowano w oryginalnej metodzie modyfikowania programów napisanych w Javie, bez jakichkolwiek zmian w kodzie źródłowym, i podejście to oraz praktyczne przykłady zastosowania opisano w pracach [H2]i [H3]. Metoda modyfikacji aspektowej została zastosowana w różnych typach aplikacji napisanych w Javie oraz w różnych częściach tych aplikacji.

Powszechne i konieczne w procesie utrzymania oprogramowania jest wprowadzanie poprawek czy zmian w funkcjonowaniu oprogramowania. Niestety często nie ma dostępu do kodu źródłowego i dokumentacji. W pracy [H2] zaproponowano **nową** metodę pozwalającą na wprowadzanie zmian w programach, napisanych w Javie, dla których **nie ma źródeł kodu ani dokumentacji**. W tym celu zastosowano programowanie aspektowe [Kic97]. Aspekty są wykorzystywane do śledzenia działania programu oraz do wprowadzenia modyfikacji do kodu.

Śledzenie aspektowe polega na wpleceniu w kod źródłowy śledzonego programu (lub jego skompilowanej wersji) aspektów zawierających kod służący do śledzenia. Może to być kod zawierający instrukcje logowania lub wysyłający komunikaty informujące o przebiegu działania programu. W wyniku wplatania, program poddawany śledzeniu zostaje więc wzbogacony o dodatkowe elementy umożliwiające jego śledzenie. Bardzo ważny w tej metodzie jest brak konieczności ręcznego dodawania komunikatów lub kodu logującego do programu. Elementy te są dołączane automatycznie za pomocą deklaratywnie tworzonych aspektów. Ważna jest również możliwość wykorzystania tej metody przy braku kodu źródłowego.

Modyfikacja aspektowa polega na utworzeniu aspektów wplatanych w kod źródłowy programu lub w sam skompilowany program poddawany modyfikacji. Aspekty mogą albo modyfikować istniejącą funkcjonalność albo wprowadzać nową. Metoda ta bazuje na zupełnie innym podejściu niż metody powszechnie używane, w których dodanie lub modyfikacja funkcjonalności następuje poprzez modyfikację kodu programu (źródłowego lub ewentualnie

skompilowanego). Należy zauważyć, ze wprowadzane aspekty są zwykłym kodem języka programowania używanego do tworzenia programu, wzbogaconym o dodatkowe elementy związane z programowaniem aspektowym. Metoda aspektowej modyfikacji programów działa w dużej mierze na zasadzie czarnej skrzynki (ang. black box). Nie jest bowiem ważne dla tej metody jak wygląda program poddawany modyfikacji. W pracy [H2] pokazano przykład modyfikacji interfejsu graficznego programu `ClassEditor` [Class] którą wykonano nie korzystając z jego kodu źródłowego ani z dokumentacji. Program `ClassEditor` jest dostępny w SorceForge.net [SFNet] na zasadzie licencji LPGL (GNU Lesser Public License) pozwalającej na jego modyfikację. Celem wprowadzonej modyfikacji był dodanie informacji o nazwie i wielkości pliku zawierającego edytowana klasę. Należało także dodać nową sekcję `File information` na ekranie programu i uaktualniać zawarte w niej informacje po każdej zmianie wykonanej przez użytkownika.

W pracy [H3] przedstawiono natomiast zastosowanie modyfikacji aspektowej do aplikacji typu Enterprise Application Archive (EAR) [EAR]. Struktura aplikacji tego typu jest złożona, może zawierać komponenty różnych typów (np. archiwa, biblioteki, aplikacje). W pracy [H3] pokazano modyfikację aplikacji `timetowork` [timet], dostępnej na zasadzie licencji BSD (Berkeley Software Distribution License – jedna z licencji zgodnych z zasadami wolnego oprogramowania), pozwalającej na modyfikacje kodu i dostępnej na SourceForge.net [SFNet] ale bez kodu źródłowego. Aplikacja `timetowork` zawiera dwa komponenty aplikację WAR (Web Application Archive) [WAR] i komponent EJB [EJB]. W aplikacji `timetowork`, służącej do zarządzania czasem dla różnych czynności wprowadzanych przez użytkownika, została dodana kontrola semantyczna wybranego pola - liczby godzin. Dodano sprawdzenie czy liczba godzin wprowadzona przez użytkownika nie przekracza 24. Tego typu kontroli aplikacja ta nie zawierała. Modyfikowany komponet EJB nie miał własnego interfejsu. GIU aplikacji `timetowork` było zrealizowane w komponencie WAR toteż pełne wprowadzenie modyfikacji wymagało modyfikacji w obu komponentach (EJB i WAR).

Śledzenie i modyfikacja aspektowa zostały także wykorzystane do zmian w logice biznesowej aplikacji typu WAR o nazwie `Minpo` [Minpo] co opisano w [Blu08] oraz aplikacji konsolowej `JMathLib` [JMath] przedstawionej w [BiB07].

Do celów śledzenia i modyfikacji aspektowej zostało zbudowane po moim kierownictwem specjalne narzędzie [BiB07]. Narzędzie to wplata aspekty w skompilowane programy w Javie, umożliwia śledzenie wykonania programu z aspektami, pozwala na filtrowanie wysyłanych komunikatów np. dotyczące pakietów, klas, metod, argumentów. Narzędzie wykorzystano także w dalszych badaniach częściowo opisanych w sekcji 3.4.2.3 (praca [H4]).

W pracach [H2] i [H3] pokazano jak w istotny i skuteczny sposób można poprawić utrzymywalność oprogramowania (ang. maintainability) ale także na zmianę funkcjonalności, poprawienia łatwości użycia, przenośność (jakość wewnętrzna i zewnętrzna) oraz użyteczności i łatwości użycia (jakość użyteczna).

3.4.2.3. Wykorzystanie algorytmów odkrywania wiedzy [H4, H5, H6, H7]

W okresie 2010-2014 prowadziłam badania dotyczące zastosowania algorytmów wydobywania wiedzy w celu poprawy istotnych kryteriów jakości oprogramowania takich jak np. niezawodność, użyteczność, bezpieczeństwo, poufność. Badania dotyczące zachowania użytkowników prowadziłam dla systemów webowych a dotyczące wykrywania anomalii dla systemów SOA (ang. Service Oriented Architecture – SOA) [SOA-1, SOA-2]. Najważniejsze wyniki tych badań przedstawiono w pracach: [H4, H5, H6, H7].

W celu poprawy użyteczności, łatwości użycia czy też funkcjonalności oprogramowania istotne jest badanie zachowania użytkowników. Na konferencji 14th International Conference

KES (2010) zaprezentowałam w pracy [H4] oryginalny system zbierający informacje o zachowaniu użytkowników, nazwany ELM (Event Logger Manager) a następnie za pomocą wybranych algorytmów odkrywania wiedzy wydobywający informacje o zachowaniu użytkownika. Wiedza ta może być następnie wykorzystana do personalizacji usług (ulepszenia funkcji sytemu czyli użyteczności, łatwości użycia) oraz do poprawy wydajności serwera. Systemy analizy zachowania użytkowników (ang. Web Usage Mining - WUM) istniejące w okresie podejmowania pracy i działające obecnie są związane z jedną aplikacją czy portalem internetowym np. Google, Amazon, Yahoo czy e-Bay. To co wyróżnia system ELM prezentowany w [H4] jest możliwość dołączenia go do dowolnego systemu napisanego w Javie, nawet jeśli nie ma dostępu do jego kodu źródłowego za pomocą zaproponowanej w [H2] metody modyfikacji aspektowej. Wymagany do dołączenia do aplikacji webowej kod aspektowy nie jest duży (zawiera określenie punktów przecięć czyli wplecenia kodu). Poprzez zmianę aspektów można określać, modyfikować gromadzone dane. Można monitorować i usprawniać proces wydobywania danych, określać filtry dla danych. Można dodać, pisząc aspekt, własny kod algorytmu wydobywania wiedzy. Kolejną cechą wyróżniającą system ELM jest możliwość gromadzenia danych z wielu aplikacji w jednej bazie. Analiza takich danych może pozwolić na szerszą analizę zachowania użytkowników. W pracy [H4] opisano rezultaty badań wykonanych na przykładowym systemie – aplikacji `jpetstore` [JPet] wykonanej w technologii JavaEE dostępnej na SorceForge.net [SFNet] gdyż nie udało się badań przeprowadzić na rzeczywistej aplikacji webowej. Wykonano szereg eksperymentów, w niektórych celowo wprowadzano pewne przypadki badając zdolność algorytmów odkrywania wiedzy do ich wykrycia. Dane z interakcji użytkownika z aplikacją `jpetstore` były powielane i losowo mieszane by symulować wielu użytkowników. Technika k- krotnej walidacji krzyżowej (ang. k-fold cross-validation) [Wit05] została użyta do podziału na dane trenujące i testujące. Badano algorytmy klasyfikacyjne ID3, C4.5, PART (na podstawie [Wit05]). Najlepsze rezultaty w wykrywaniu prawidłowości w zachowaniu użytkowników uzyskano dla algorytmów C4.5 i PART. Wykonano także eksperymenty dla algorytmów tworzących reguły asocjacyjne (ang. association-rule learners) Apriori, Tertius i Predictive Apriori z biblioteki weka [weka]. Algorytmy te także wykrywały oczywiste regularności w zachowaniu użytkowników, także te, które celowo wprowadzono.

W pracy [H4] opisano narzędzie i metodę zbierania informacji o zachowaniu użytkownika co jest przydatne przy poprawianiu jakości użytecznej (użyteczność, łatwość użycia) oraz funkcjonalności i użyteczności (jakość zewnętrzna i wewnętrzna).

Algorytmy odkrywania wiedzy mogą być także wykorzystane do detekcji różnego rodzaju anomalii, błędów w działaniu systemu, mogą wskazywać na zagrożenia poufności, niezawodności i bezpieczeństwa systemu. Obecnie coraz więcej systemów realizowanych jest w architekturze zorientowanej serwisowo (ang. Service Oriented Architecture – SOA) [SOA-1], [SOA-2]. System w architekturze SOA jest zbiorem usług czy systemów dostarczających usługi, często z różnych dziedzin poprzez sieć. W tego typu architekturze trudno jest wykryć anomalie, braki w bezpieczeństwie czy poufności.

W pracy [H5] przedstawiono system o architekturze SOA, zaprojektowany i zbudowany specjalnie do celów badania przydatności wybranych algorytmów sztucznej inteligencji do wykrywania różnego rodzaju anomalii gdyż nie można było prowadzić badań na „pracującym”, rzeczywistym systemie. W systemie tym zaimplementowano cztery algorytmy odkrywania wiedzy: grupowania danych k-means [Mun07], sieci Kohonen’a [Koh90], wzorców wyłaniających (ang. emerging patterns) [Don99] oraz statystyki Chi-Square [Mas00]. Wprowadzane były różne typy anomalii i badano jaka jest efektywność algorytmów w wykrywaniu tych anomalii. Podstawowymi wartościami mierzonymi w poszczególnych eksperymentach były:

- FP – liczba przykładów fałszywie zakwalifikowanych jako anomalie (ang. false positive),
- FN – liczba przykładów fałszywie niezakwalifikowanych jako anomalie (ang. false negative),
- TP – liczba przykładów prawidłowo zakwalifikowanych jako anomalie (ang. true positive),
- TN – liczba przykładów prawidłowo zakwalifikowanych jako zachowanie normalne (ang. true negative).

Najlepszy detektor anomalii ma zapewnić najlepsze wyniki, czyli jak najmniejszą liczbę błędów. W celu precyzyjnego zdefiniowania odpowiednich kryteriów wprowadza się miary jakości. Często są stosowane miary: specyficzność (także swoistość) (ang. specificity) oraz czułość (ang. sensitivity). Czułość definiujemy jako: stosunek TP do sumy TP i FN. Specyficzność jest zdefiniowana jako stosunek TN do sumy TN i FP.

W pracy [H5], przebadano najprostszy przypadek, w którym zmianie ulega częstotliwość wywołania wyłącznie jednego serwisu. Jest to bardzo typowa anomalia występująca w systemie o architekturze zorientowanej serwisowo. Mimo to wiedza o zaistnieniu takiej anomalii może okazać się ważna dla zespołu utrzymaniowego gdyż może świadczyć o poważnej nieprawidłowości, zaburzeniu niezawodności czy bezpieczeństwa. Jej zaistnienie może świadczyć o błędnej konfiguracji, błędzie w programie lub o konieczności zmiany przydziału zasobów. Najlepiej anomalie te wykrywał algorytm wzorców wyłaniających, dobre rezultaty otrzymano także dla algorytmu sieci Kohonen'a a najgorsze dla algorytmu grupowania danych *k*-means.

W pracy [H6], zbadano zdolność algorytmów do wykrywania nieużywanych funkcji. Pewne funkcje mogą nie być używane np. ze względu na błędy w logice biznesowej czy algorytmach trasowania (ang. routing) czy też wynikać ze zmian użyteczności. Wykrycie nieużywanej funkcji, jeśli jest spowodowane zmianami użyteczności, może pozwolić na alokacje w tym klastrze innej funkcji/ usługi i poprawę wydajności całego systemu. Tego typu anomalię najlepiej wykrywały algorytmy sieci Kohonen'a oraz algorytm grupowania danych *k*-means, który nie sprawdził się w przypadku zmiany częstotliwości jednej usługi. Algorytmy te wymagały jednak dużej liczby danych trenujących.

Badania przydatności algorytmów uczących się w systemach zorientowanych serwisowo kontynuowałam i w 2014 została opublikowana praca [H7]. W pracy tej przedstawiono badanie wykrywania zmiany częstotliwości grupy serwisów. Tego typu sytuacja może wystąpić w rzeczywistym systemie np. w przypadku zmiany charakteru danych wejściowych, zmian trasowania a jej wykrycie może służyć np. wykryciu problemów z bezpieczeństwem systemu czy optymalizacji zasobów. Algorytm wzorców wyłaniających (najlepszy w wykrywaniu anomalii dotyczącej pojedynczego serwisu) nadal był najlepszy biorąc pod uwagę specyficzność i czułość. Poprawiły się rezultaty algorytmu grupowania danych *k*-means (najsłabszego przy wykrywaniu anomalii dotyczącej pojedynczego serwisu). Ważne jest by nie tylko wykryta była anomalia ale także by wiadomo było jakie serwisy w niej uczestniczą. Bardzo przydatne informacje, z nazwami serwisów uczestniczących w anomalii, są produkowane przez algorytmy wzorców wyłaniających i sieci Kohonen'a. Algorytm grupowania danych k-means nie daje takich możliwości.

W pracach [H5, H6, H7] pokazano, że algorytmy uczące się w systemach o organizacji serwisowej pozwalają na wykrycie braków w:

- bezpieczeństwie i poufności, niezgodności z kontekstem (poprawa jakości użytecznej) oraz
- poufności, zgodność technicznej, niezawodności, użyteczności, oraz mogą przyczynić się do poprawy wydajności (poprawa jakości zewnętrznej i wewnętrznej).

3.4.2.4. Testowanie

Jedną z kluczowych czynności w procesie produkcji oprogramowania wysokiej jakości jest testowanie np.: [Sur14], [TiF14], [Tia05]. Metody testowania rozwijały się razem z powstającymi językami programowania, typami aplikacji czy środowiskami produkcji oprogramowania. Bibliografia dotycząca testowania jest niezwykle rozległa i zawiera wiele interesujących pozycji pochodzących sprzed wielu lat np. [Her76], [Bez84] ale i obecnie publikowane są nowe prace np. [RLR14], [TiF14].

Podstawowe podejścia do testowania ([TiF14], [Bez84], [Wis09]) to testowanie funkcjonalne (ang. black box) odbywające się na podstawie specyfikacji funkcjonalnej testowanej jednostki, oraz strukturalne (ang. white box), w którym wykorzystuje się znajomość kodu do przygotowania przypadków testowych. Metody testowania strukturalnego można podzielić na dwie grupy: podejścia, w których dąży się do pokrycia przepływu sterowania (ang. control flow coverage) np. [Woo06], [Mal06] oraz pokrycia przepływu danych (ang. data flow coverage) [Ara14]. Na innych zasadach opiera się testowanie mutacyjne. W latach 2006-2014 prowadziłam badania dotyczące różnych typów testowania ze szczególnym uwzględnieniem testowania w oparciu o przepływ danych, czemu poświęcone są prace [H8, H9] oraz testowania mutacyjnego programów napisanych w Javie [Java] (prace [H9, H10]). Testowanie oparte o przepływ danych ze względu na duży wkład pracy w jego wykonanie jest rzadko przedmiotem badań i nie jest stosowane w przemyśle. Celem prowadzonych prac było sprawdzenie jego przydatności oraz wykonanie narzędzia umożliwiającego stosowanie tej metody. Wykonano szereg eksperymentów a także zrealizowano narzędzie umożliwiające ich przeprowadzenie. Prace badawcze dotyczące testowania mutacyjnego programów napisanych w Javie dotyczyły obniżenia wysiłku związanego z tym testowaniem przy zachowaniu akceptowalnej jakości testowania.

3.4.2.4.1.Testowanie pokrycia przepływu danych [H8]

Idea testowania w oparciu o pokrycie przepływu danych została zaproponowana przez Herman'a w 1976 roku [Her76], następnie była rozwijana w wielu pracach np. Laski i Korel [Las83], Rapps i Weyuker [Rap85], Harold i Soffa [Har89]. Mimo że eksperymenty pokazują np. [Hwa99], [Hut94] wysoką efektywność tego typu testowania, bardzo dobrą lokalizację błędów np. [San09], metoda ta jest mało znana i rzadko stosowana. Jedną z przyczyn małej popularności jest brak narzędzi dostosowanych do obecnych środowisk i języków [H8].

W pracy [H8] pokazano rozszerzenie (ang. plug-in) systemu Eclipse, nazwane DFC (ang. Data Flow Coverage), zaprojektowane i zrealizowane pod moim kierunkiem, umożliwiające testowanie programów napisanych w Javie metoda pokrycia przepływu danych. W owym czasie istniało tylko jedno podobne narzędzie o nazwie JaBUTi [JaBUT], udostępnione na uniwersytecie w Brazylii.

W testowaniu opartym o pokrycie przepływu danych przypadki testowe opracowuje się w oparciu o zależności między danymi. Bada się różne ścieżki od miejsca w którym dana/zmienna została *zdefiniowana* (ang. def.) do miejsca, w którym jest *używana* (ang. use). Stosuje się różne kryteria wyboru ścieżek *definicja-użycie* (ang. def-use) np. pokrycie wszystkich ścieżek def-use. W pracy [H8] pokazano testowanie przykładowej metody napisanej w Javie za pomocą wtyczki DFC. Pokazano, na przykładzie, że testowanie przepływu danych może wykryć błędy nie wykrywane w testowaniu funkcjonalnym.
Narzędzie wspomagającego testowanie metoda pokrycia przepływu danych DFC zaprezentowałam na konferencji SET w 2009 a praca została opublikowana w LNCS w 2012 [Blue12].

3.4.2.4.2. Porównanie testowania mutacyjnego i testowania metodą pokrycia przepływu danych [H9]

Innym podejściem do testowania jest technika wstrzykiwania błędów (ang. fault injection) np. [Voa97]. Do testowanego programu wprowadzane są celowe błędy, których wykrycie ma umożliwić wykrycie błędów istniejących już wcześniej [DBG11, Wis09]. Zauważono, że wiele błędów popełnianych przez programistów to proste pomyłki np. niewłaściwy operator, wywołanie niewłaściwej metody lub funkcji. Na tej obserwacji bazuje testowanie mutacyjne, zaproponowane przez Liptona w 1971 wg. [Mat94]. Do programu celowo wprowadza się drobną zmianę np. zmieniając operator, zamieniając zmienną. Taki zmieniony program nazywa się mutantem. Mutanty wykonuje się podając przygotowane dane testowe. Przypadki testowe powinny skutkować innym stanem dla oryginalnego programu i mutanta. Jeśli test wykryje wprowadzoną zmianę oznacza to że mutant został wykryty (ang. killed). Niektóre mutanty dla dowolnego zestawu danych wejściowych dają takie same wyniki działania jak program oryginalny i nie mogą zostać wykryte, nazywane są mutantami równoważnymi (ang. equivalent mutant) [MOT14].

Od wprowadzenia, testowanie mutacyjne i testowanie metodą pokrycia przepływu danych były uznane jako potencjalnie efektywne ale niewiele prac dotyczyło badania ich efektywności i porównania. W latach 1993-1996 przeprowadzono kilka eksperymentów porównujących testowania mutacyjne i testowanie pokrycia przepływu danych dla programów napisanych w językach: Fortran [MWo94], C [OPa96], i Pascal [Fra97]. Przeprowadzono eksperyment (2009) porównujący testowanie mutacyjne i testowanie pokryciem przepływu danych dla wybranych programów napisanych w Javie a jego rezultaty opisano w pracy [H9] (złożonej w 2010 a opublikowanej w 2011). Celem badań było sprawdzenie jaką efektywność w testowaniu mutacyjnym można uzyskać za pomocą testów przygotowanych dla metody pokrycia przepływu danych i na odwrót. W chwili rozpoczynania badań nie było w literaturze informacji o podobnych badaniach. W pracy [H9] pokazano, że testowanie metodą pokrycia przepływu danych dla testów opracowanych dla testowania mutacyjnego daje bardzo dobre rezultaty. Dla testów przygotowanych dla narzędzia mutacyjnego MuClipse [Mucli] średnio (dla wszystkich programów będących przedmiotem eksperymentu) ponad 96 % par *def-use* zostało pokrytych. Dla kilku metod pokrycie było nawet pełne. Tak dobre pokrycie wynika z bardzo dużej liczby testów mutacyjnych. Zastosowanie testów pokrycia przepływu danych do wykrycia mutantów nie dało aż tak dobrych wyników. Uzyskano średnio 88% wskaźnik mutacji, czyli stosunek liczby zabitych mutantów do liczby nierównoważnych mutantów (ang. mutation score). W 2010 roku opublikowano rezultaty innego eksperymentu ale też dotyczącego efektywności testowania mutacyjnego programów napisanych w Javie [Mad10].

W pracy [H9] wykazano, że testowanie mutacyjne jest efektywniejsze niż testowanie oparte o pokrycie przepływu danych, jednak jest ono bardziej kosztowne biorąc pod uwagę czas i wysiłek potrzebny do jego przeprowadzenia. Istotne jest zmniejszenie wysiłku obliczeniowego, który wynika z dużej liczby wykonywanych testów oraz kosztów „ludzkich", wynikających z konieczności „ręcznej identyfikacji" mutantów równoważnych. Dalsze badania dotyczyły tej właśnie tematyki.

3.4.2.4.3. Zmniejszenie wysiłku związanego z testowaniem mutacyjnym [H10]

Czas testowania mutacyjnego można skrócić przez równoległe wykonywanie testów jak zaproponowali Mateo i Usaola w 2013 [Mat13] lub redukując liczbę testów. Zmniejszenie liczby mutantów zmniejszy zarówno koszty obliczeniowe jak i „ludzkie". Istotne jest by tak zredukować liczbę mutantów by zachować dobrą jakość testowania. Próbkowanie mutantów (ang. mutant sampling) było zaproponowane w 1980 roku przez Acree [Acr80] i Budd'a

[Bud80] jednak konkretne techniki redukcji zostały zaproponowane przez Mathur i Wong [Mat95] w 1995 roku. Zaproponowali oni losową redukcje x% mutantów lub wybór pewnych operatorów mutacyjnych (ang. constrained mutation). Prace dotyczące wyboru efektywnych operatorów dla Fortranu prowadzili np. Offutt, Rothermel i Zapf [Off96] oraz Mresa i Bottaci [Mre99]. Losową redukcją mutantów dla programów napisanych w Javie zajmowałam się w pracy [Blu13] a problemem redukcji pewnych operatorów mutujących Javy w pracy[Blu14]. Scholive, Beroulle i Robach [Sch05] zaproponowali w 2005 roku by redukować o pewien procent zbiory testów przygotowane dla sprzętu. Technikę tę wykorzystałam w kolejnych eksperymentach dotyczących testowania mutacyjnego i opisanych w [H10].

W pracy [H10] przedstawiłam wyniki eksperymentu polegającego na redukcji od 40% do 90 % mutantów wygenerowanych dla poszczególnych operatorów mutacyjnych (programy w Javie, narzędzie mutacyjne Muclipse) stosując skok 10%. Okazało się, że 40 % mutantów wystarcza by uzyskać współczynnik wykrycia mutantów powyżej 95% i poniżej 1% degradacji w pokryciu kodu programu testami. Wyniki te są istotnie lepsze niż dla tych samych programów i narzędzia mutacyjnego wyniki uzyskane przy losowej eliminacji mutantów (60% mutantów) opisanej w pracy [Blu13]. Mimo że eksperymenty wykonywane były w innym środowisku i dla innego języka programowania, jego rezultaty potwierdziły obserwacje Scholive, Beroulle i Robach'a [Sch05], że redukcje selektywne są bardziej efektywne niż losowe.

Prace dotyczące testowania [H8], [H9] i [H10] pozwalają na polepszenie zgodności technicznej, niezawodności (jakość zewnętrzna i wewnętrzna) oraz bezpieczeństwa (jakość użyteczna) oprogramowania.

3.4.3 Podsumowanie

Aktywność naukowa przedstawiona w niniejszym opracowaniu pozwala na znaczącą poprawę kryteriów jakości oprogramowania obejmujących wszystkie cechy jakości oprogramowania zgodnie z normą ISO 25000:2005 (co pokazano w tabeli 1). W szczególności wkład przedstawionych publikacji w dziedzinę poprawy jakości oprogramowania obejmuje:

- Nową, skuteczną i efektywną metodę zmiany funkcjonalności oprogramowania za pomocą programowania aspektowego. Metoda ta umożliwia modyfikację oprogramowania bez dostępu do kodu źródłowego (2008-2010), prace [H2]i [H3].
- Obliczanie metryk obiektowych zintegrowane z narzędziem CASE do projektowania oprogramowania (2001, praca [H1])
- Nową metrykę obiektową dającą wskazania o pracochłonności testowania (praca [H1], 2001).
- Opracowanie koncepcji opartej na modyfikacji aspektowej i wykonanie przykładowego narzędzia do zbierania danych z interakcji użytkownika oraz wnioskowania o jego potrzebach w oparciu o algorytmy sztucznej inteligencji (2009-2010) ([H4]).
- Badanie przydatności algorytmów wydobywanie wiedzy do wykrywania anomalii w systemach o architekturze serwisowej (2012-2014) ([H6], [H7]).
- Badanie testowania przepływu danych dla programów w Javie (2009-2012) ([H8])
- Porównanie testowania mutacyjnego i opartego o pokrycie przepływu danych dla programów w Javie (2009-2011) ([H9]).
- Badania dotyczące redukcji liczby mutantów dla programów napisanych w Javie, których celem jest zmniejszenie pracochłonności testowania mutacyjnego (2011-2014), ([H10]).

Przy przygotowaniu prac [H1] – [H10] powstało szereg narzędzi wspomagających produkcje i utrzymanie oprogramowania i przyczyniających się do poprawy jakości oprogramowania. Kierowałam projektem i wykonaniem następujących narzędzi wykorzystanych w pracach stanowiących cykl:

- Rozszerzenia komercyjnych narzędzi CASE (Paradigm-2000, Select Enterprice-2003, Rose-2004, Software Architect- 2005) obliczające metryki obiektowe [H1], [Blu04].
- Narzędzie wspomagające śledzenie i modyfikację aspektową (2007) [H2-H3].
- System symulacji i pomiarów anomalii w systemach o architekturze zorientowanej serwisowo, wykrywanie anomalii algorytmami sztucznej inteligencji (2011) [H5-H7].
- Rozszerzenie (ang. plug in) DFC do systemu Eclipse do testowania oprogramowania obiektowego metodą badania pokrycia przepływu danych (2009) [H8] .
- System odkrywania potrzeb użytkownika na podstawie jego interakcji z aplikacją internetową (ELM) - (2008) [H4].

3.5. Literatura dodatkowa

[Acr80] Acree A.T.: On mutation, Ph.D. thesis, Georgia Institute of Technology, Atlanta, 1980

[Ara14] de Araujo A., Chaim M. L.: Data-flow Testing in the Large, Proc. Int. Conf. on Soft. Testing, Verification. and Validation, str. 81-90, (2014), DOI 10.1109/ICST.2014.19

[Bak11] Bakota T., Hegedus P., Kortvelyesi P., Ferenc R., Gyimothy T.: A probabilistic software quality model, 27th Int. Conf. on Soft. Maintenace (ICSM), 2011, str. 243-252

[BTD14] Bartoszuk C., Timoszuk G., Dabrowski R., Stencel K.: On Visual Assessment of Software Quality, e-Informatica Software Engineering Journal, vol. 8, no 1, 2014, str. 7–26, DOI 10.5277/e-Inf140101

[Beg99] Begier B.: Inżynieria Oprogramowania - Problematyka Jakości, WPP, 1999

[Beg03] Begier B.: Ocena jakości wyrobu programowego przez użytkowników, w: Problemy i metody inżynierii oprogramowania, Huzar Z., Mazur Z. (eds.), str. 417-432, WNT, Warszawa 2003

[BeJ90] Belli F., JędrzejowiczP. : Fault - Tolerat Programs and Their Realiability, *IEEE Trans. on Reliability*, vol. 39, no 2, str. 184-192, 1990

[Bez84] Bezier B.: Software System Testing and Quality Assurance, Van Nostrad Rheinhold, New York, 1984

[BiB07] Billewicz K., Bluemke I.: Aspekty w śledzeniu i modyfikowaniu skompilowanych programów, rozdz. w: *„Zwinność i dyscyplina w inżynierii oprogramowania*, A. Jaszkiewicz, [et.al] (red.), Wydawnictwo Nakom Poznań, 2007, str. 23-34, ISBN 978-83-89529-45-9

[Bil07] Bilski E., Dubielewicz I.: Cykl życia oprogramowania modele, procesy, jakość w normach ISO, Oficyna Wyd. Politechniki Opolskiej , ISBN 9788374933360, 2007

[Blu01] Bluemke I.: Evaluation of class testing complexity with object metrics, w: Proc. of Int. Conf. on Internet and Multimedia Systems and Applications, IMSA 2001, August 13-16 Honolulu, Hawaii, USA, str. 357-361

[Blu04] Bluemke I., Zając P.; Evaluation of object metrics in a CASE, Software Engineering Conference 2004, IASTED, 17-19 Feb 2004, Innsbruck, str. 626-631

[Blu08] Bluemke I., Billewicz K.: Aspects modification in business logic of compiled Java programs, IEEE First Int. Conf. on Information Technologies, Gdansk, Poland, May 2008, str. 409-412, DOI:10.1109/INFTECH.2008.4621670

[Blu12] Bluemke I., Rembiszewski A: Dataflow Testing of Java Programs with DFC, w: Advances in Software Engineering Techniques, Szmuc T., Szpyrka M., Zendulka J. (red.), LNCS, vol. 7054, 2012, Springer, str. 215-228, DOI:10.1007/978-3-642-28038-2_17

[Blu13] Bluemke I., Kulesza K.: Reduction of Computational Cost in Mutation Testing by Sampling Mutants, w: New Results in Dependability and Computer Systems, Proc. of the 8th Int. Conf. on Dependability and Complex Systems DepCoS-RELCOMEX, Zamojski W. [et al.] (eds.), *Advances in Intelligent Systems and Computing*, vol. 224, 2013, Springer, str. 41-51, DOI:10.1007/978-3-319-00945-2_4

[Blu14] Bluemke I., Kulesza K.: Reductions of operators in Java mutation testing, DepCoS-RELCOMEX 2014, w *Advances in Intelligent and Soft Computing*, Springer, 2014, vol. 286, str. 93-102, DOI:10.1007/978-3-319-07013-1_9

[Boe78] Boehm B., Brown J., Lipow M., MacCleod G.: Characteristics of software quality, NY, American Elsevier, 1978

[Bro78] Broy M.: Requirements Engineering as a Key to Holistic Software Quality, w *Computer and Information Sciences* – ISCIS 2006, vol. 4263, LNCS, str. 24–34.

[Bud80] Budd T.A: Mutation analysis of program test data, Ph.D. thesis, Yale University, New Haven, Connecticut, 1980.

[McC77] McCall J., Richards P., Walters G.: Factors in software quality, Griffiths Air Force Base, NY, Rome Air Development Center Air Force Systems Command, 1977

[ChK94] Chidamber S.R., Kemerer C.F.: A metrics suite for object oriented design, *IEEE Trans. on Soft. Eng*., vol. 20, no 6, 1994, str. 476-492

[Class] ClassEditor – http:sourceforge.net/projects/classeditor

[Cot06] Côté M.A., Suryn W., Georgiadou E.: Software Quality Model Requirements for Software Quality Engineering, Software Quality Management & INSPIRE Conference (BSI), 2006
[Dav11] Davis A. M.: bibliografia dotycząca inżynierii wymagań: http://www.reqbib.com/ (prowadzona do 2011)
[Dei07] Deissenboeck F., Wagner S., Pizka M., Teuchert S., Girard J.F.: Activity-Based Quality Model for Maintainability, w Proc. 23rd Int. Conf. on Software Maintenance (ICSM 2007), IEEE CS Press, 2007
[Dij76] Dijkstra E.: A Discipline of Programming, 1976, ISBN-13: 978-0132158718
[DBG11] Domínguez-Jiménez J.J., Estero-Botaro A., García-Domínguez A, Medina-Bulo I.: Evolutionary mutation testing, *Information and Software Technology*, vol. 53, str. 1108-1123, Elsevier, 2011
[Don99] Dong G., Li J.: Efficient Mining of Emerging Patterns: Discovering Trends and Differences, KDD '99, Proc. of the fifth ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining, str. 43-52 , ACM, NY, USA, 1999, ISBN:1-58113-143-7, DOI:10.1145/312129.312191
[Dro95] Dromey R.G.: A model for software product quality, *IEEE Trans. on Soft. Eng*., vol. 21, no 2, 1995, str. 146-162
[Dym00] Dymek D.:Zarządzanie jakością oprogramowania komputerowego, Akademia Ekonomiczna w Krakowie 2000
[EAR] http://docs.sun.com/app/docs/doc/8193875/6n62klump?a=view#jesgl-aoh, 2008
[Eck08] Eckroth J., Amoussou G.-A.: Improving Software Quality from the Requirements Specification, in SoD '07 Proc. of the 2007 Symposium on Science of Design, str. 38-39, DOI: 10.1145/1496630.1496651
[EJB] http://docs.sun.com/app/docs/doc/8193875/6n62klump?a=view#jesgl-bxx, 2008
[Fra97] Frankl P.G., Weiss S.N., Hu C.: All-uses versus mutation testing: An experimental comparison of effectiveness, *Journal of Systems and Software*, vol. 38, str. 235–253, 1997
[Gar87] Garvin D.A.: Competing in the Eight Dimensions of Quality, Harvard Business Review, Sept.-Oct. 1987
[Glin09] Glinz et al.: Report on the Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ'09) w: ACM SIGSOFT Software Engineering Notes, vol. 34, no.5, str. 40-45, DOI: 10.1145/1598732.1598759
[GoŁ13] Górski J., Łukasiewicz K.: Towards Agile Development of Critical Software, *Software Engineering for Resilient Systems*, LNCS vol. 8166, str. 48-55, 2013
[GJD12] Gross A, Jurkiewicz J., Doerr J., Nawrocki J.: Investigating the Usefulness of Notations in the Context of Requirements Engineering, EmpiRE 2012, Chicago, Illinois, USA, str. 9-16
[GSc07] Gordon Schulmeyer G. (editor): Handbook of Software Quality Assurance, ISBN-13: 978-1450421041, ISBN-10: 1450421040, ed. 4, 2007
[Har89] Harold M.J., Soffa M.L.: Inter-procedural data flow testing, Proc. of the Third Testing, Analysis, and Verification Symposium, str. 158 – 167, 1989
[Her76] Herman P. M.: A data flow analysis approach to program testing, *Australian Computer Journal*, vol. 8, no. 3, str. 92–96, 1976.
[Hoff1] Hoffman R.: Czym jest jakość oprogramowania, http://softlex.org/materialy/Radoslaw_Hofman.pdf
[Hoff2] Hoffman R.: Modele jakości oprogramowania - historia i perspektywy, www.teycom.pl\docs
[Hut94] Hutchins M., Foster H., Goradia T., Ostrand T.: Experiments of the effectiveness of dataflow- and control-flow-based test adequacy criteria, Proc. of the 16th Int. Conf. on Soft. Eng., ICSE '94, str. 191–200, 1994
[Hwa99] Mei-Hwa Chen, Kao H.M.: Testing Object-Oriented Programs An Integrated Approach, Proc. of the 10th Int. Symposium on Software Reliability Engineering, 1999, str. 73-83, DOI: 10.1109/ISSRE.1999.809312
[ISO91] JTC1/SC7, Information Technology – Software Product Quality, International Standardization Organization, 1991
[ISO01] JTC1/SC7, Software engineering - Product quality, International Standardization Organization, 2001
[ISO05] ISO/IEC 25000:2005, JTC1/SC7, Software Engineering - Software product Quality Requirements and Evaluation (SQuaRE), International Standardization Organization, 2005
[JaBUT] http://jabuti.incubadora.fapesp.br/
[Java] http://www.oracle.com/technetwork/java/index.html
[JMath] JMathLib – http://sourceforge.net/projects/mathlib
[JoE94] Jorgensen P.C., Erickson C.: Object-oriented integration testing, *Communications of the ACM*, vol. 37, no 9, 1994, str. 31-38
[JPet] JPetStore, http://sourceforge.net/projects/ibatisjpetstore/
[Kan06] Kan S.: Metryki i modele w inżynierii jakości oprogramowania, Wydawnictwo Naukowe PWN, 2006
[KGK09] Kumar A., Grover P.S., Kumar R.: A Quantitative Evaluation of Aspect-Oriented Software Quality Model (AOSQUAMO), *ACM SIGSOFT Software Engineering Notes*, vol 34, no 5, 2012, DOI: 10.1145/1598732.1598736

[Kho07] Khoshgoftaar T.M., Yi Liu: A multi-objective software quality classification model using genetic programming, IEEE Transactions on Reliability, vol. 56, no 2, June 2007, str. 237-245, DOI: 101109/TR.2007.896763
[Kic97] Kiczales G., Lamping J., Mendhekar A., Maeda Ch., Lopes C., Loingtier J., Irwin J.: Aspect-Oriented Programming, Proc. European Conf. on Object-Oriented Programming, vol. 1241, Springer, 1997, str. 220–242.
[KiT94] Kirami S., Tsai W.T.: Method sequence specification and verification of classes, *Journal of Object-Oriented Programming*, October 1994, str. 28-38
[Kit96] Kitchencham B., Pfleger S.L.: Software quality: the elusive target, *IEEE Software*, vol. 13, no 1, str. 12-21, 1996
[Kob03] Kobyliński A.: ISO/IEC 9126 - analiza modelu jakości produktów programowych, w: Systemy Wspomagania Organizacji 2003, Porębska-Miąc T., Sroka H. (red.), Prace Naukowe AE w Katowicach, 10, 2003.
[Kob05] Kobyliński A.: Modele jakości produktów i procesów programowych, Oficyna Wydawnicza Szkoły Głównej Handlowej, 2005
[Koh90] Kohonen T.: The self-organizing map, *Proc. IEEE*, vol. 78, no. 9, str. 1464-1480, Sept. (1990)
[KoN14] Kopczyńska S., Nawrocki J.: Using Non-functional Requirements Templates for Elicitation: A Case Study, Proceedings of RePa 2014, Karlskrona, Sweden, str. 47-54
[Kot00] Kotonya G., Sommerville I.: Requirements engineering: processes and techniques, Wiley, 2000
[Loc11] Lochmann K., Goeb A.: A unifying model for software quality, w Proc. WoSQ'11, Sept. 4, 2011, Szeged, Hungary, ACM 978-1-4503-0851-9/11/09 , str. 3-10
[Lad03] Laddad R.: AspectJ in Action X : Practical Aspect-Oriented Programming, Manning, 2003, ISBN 1-930110-93-6
[Las83] Laski J., Korel B.: A Data Flow Oriented Program Testing Strategy, *IEEE Trans. On Soft. Eng*., vol. 9, no. 3, May, str. 347- 354, 1983
[Mad10] Madeyski L.: The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment, *Information and Software Technology*, vol. 52, no. 2, Elsevier, str. 169-184, 2010
[MaK04] Madeyski L., Karwaczyński P.: Działania projakościowe w procesie wytwarzania oprogramowania, Prace Naukowe Akademii Ekonomicznej we Wrocławiu, Nowoczesne Technologie Informacyjne w Zarządzaniu, 2004
[Mal06] Malevris N., Yates D.F.: The collateral coverage of data flow criteria when branch testing, *Information and Soft. Techn.*, vol. 48, str. 676-686, 2006
[Msz07] Madeyski L., Szała Ł. : Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study, IET Softw., vol. 1, no 5, str. 180–187, 2007, DOI 0.1049/iet-sen:20060071
[Mas00] Masum S., Ye E.M., Chen Q., Noh K.: Chi-square statistical profiling for anomaly detection, w Proc. of the 2000 IEEE Workshop on Information Assurance and Security, 2000
[Mat13] Mateo P. R, Usaola M. P.: Parallel mutation testing, *Softw. Test. Verif. Reliab*, vol.23, str. 315–350, 2013.
[Mat94] Mathur A. P.: Mutation Testing, w : *Encyclopedia of Software Engineering*, Marciniak J. J. (ed.), str.707–713, 1994
[Mat95] Mathur A. P, Wong W.E.: Reducing the cost of mutation testing: an empirical study, *J. Syst. Softw*., vol. 31, no. 3 str. 185-196, 1995
[MBR09] Martens A., Brosch F., Reussner R.: Optimising Multiple Quality Criteria of Service-Oriented Software Architectures, w Proceedings of QUASOSS'09, August 25, Amsterdam, str. 25-32, 2009
[McC77] McCall J., Richards P., Walters G.: Factors in Software Quality, 1-3, RADC-TR-77-369, US Rome Air Development Center, Griffiss Air Force Base , NY 13441-5700, November 1977
[Minpo] http://sourceforge.net/projects/minpo (dostęp 2008)
[MOT14] Madeyski L., Orzeszyna W., Torkar R.: Overcoming the Equivalent Mutant Problem: A Systematic Literature Review and a Comparative Experiment of Second Order Mutation, IEEE Trans. On Soft. Eng., vol. 40, no 1, str. 23-44, 2014
[Mre99] Mresa E.S., Bottaci L.: Efficiency of mutation operators and selective mutation strategies: An empirical study, *Soft. Testing, Ver. and Rel.* , vol. 9, no 4, str. 205-232, 1999
[MWo94] Mathur A. P., Wong W.E.: An empirical comparison of data flow and mutation-based test adequacy criteria, The *J. of Soft. Test., Ver., and Rel.*, vol. 4, no. 1, str. 9-31, 1994
[Mucli] MuClipse, http://muclipse.sourceforge.net/index.php (dostęp 2012)
[Off96] Offutt J., Rothermel G., Zapf C.: An experimental determination of sufficient mutation operators, *ACM Trans. on Soft. Eng. and Methodology*, vol. 5, no 2, str. 99-118, 1996
[Mun07] Munz G., Li S., Carle G.: Traffic Anomaly Detection Using K-Means Clustering, W. Schickard Institute for Computer Science, University of Tuebingen, 2007

[Nus00] Nuseibeh B., Easterbrook S.: Requirements Engineering: A Roadmap, ICSE 2000 Future of Software Engineering, str. 35-46, ACM, NY, USA, 2000, DOI: 10.1145/336512.336523
[OPa96] J. Offutt, J. Pan, K. Tewary, T. Zhang, "An Experimental Evaluation of Data Flow and mutation Testing", *Soft. Prac. & Exp*., vol. 26, no 2, str. 165-176, 1996
[Par72] Parnas D.: On the criteria to be used in decomposing systems into modules, Communications of the ACM , vol. 15, no 12, str. 1053–58. DOI:10.1145/361598.361623, 1972
[Pfl01] Pfleeger S., Atlee J.M.: Software Engineering: Theory and Practice, (4th Edition), ISBN-10: 0136061699, 2009
[PoS14] Poth A., Sunyaev A.: ,Effective Quality Management Value- and Risk-Based Software Quality Management, IEEE Software, November/December 2014, str. 79-85
[Ral13] Ralph P.: The Illusion of Requirements in Software Development, *Requirements Engineering*, Sept. 2013, vol. 18, no 3, str. 293-296, 2013
[Rap85] Rapps S., Weyuker E.J.: Selecting software test data using data flow information, *IEEE Trans. on Soft. Eng*., vol. 11, str. 367–375, 1985
[RLR14] Rodriguez I., Llana L., Rabanal P.: A General Testability Theory: Classes, Properties, Complexity, and Testing Reductions, *IEEE Trans. on Soft. Eng*., vol. 40, no 9, 2014, str. 862–894, DOI: 10.1109/TSE.2014.2331690
[RPB12] Rashid E., Patnayak S., Bhattacherjee V.: A Survey in the Area of Machine Learning and Its Application for Software Quality Prediction, ACM SIGSOFT Software Engineering Notes, vol.37, no 5, September 2012, str. 1-7, DOI: 10.1145/2347696.2347709
[Sac10] Sacha K.: Inżynieria Oprogramowania, PWN 2010
[San09] Santelices R., Jones J. A., Yu Y., Harrold M. J.: Lightweight fault-localization using multiple coverage types, w Proc. of the 31st Int. Conf. on Soft. Eng., ICSE '09, str. 56–66, 2009
[Sch05] Scholive M., Beroulle V., Robach C.: Mutation Sampling Technique for the Generation of Structural Test Data, w In: Proc. of the Conf. on Design, Automation and Test in Europe, vol. 2, str. 1022 – 1023, 2005
[SFNet] SourceForge:http://sourceforge.net, 2008.
[Sin13] Singh B., Kannojjia S. P.: A review on software quality models, in Int Conf. on Communication Systems and Network Technologies, 2013, DOI: 10.1109/CSNT.2-13.171
[SOA-1] BPEL Standard, http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, (access July 2011)
[SOA-2] SOA manifesto: http://www.soa-manifesto.org (access July 2011)
[Sur03] Suryn W., Abran A., ISO/IEC SQuaRE. The second generation of standards for soft. product quality, IASTED 2003
[Sur14] Suryn, W.: *Software Quality Engineering: A Practitioner's Approach*, 2014, Wiley-IEEE Press eBook, DOI: 10.1002/9781118830208
[Ter13] Terzakis J.: The Impact of Requirements on Software Quality across Three Product Generations, 978-1-4673-5765-4/13/ IEEE, w RE 2013 str. 284-289
[Tia05] Tian, J.: Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement, WILEY-IEEE PRESS EBOOK, 2005, DOI: 10.1002/0471722324
[TiF14] Tilley S.; Floss B.: *Hard Problems in Software Testing: Solutions Using Testing as a Service (TaaS)*, Morgan & Claypool, DOI: 10.2200/S00587ED1V01Y201407SWE002 , 2014
[timet] http://sourceforge.net/projects/timetowork, 2008
[TuR93] Turner C.D., Robson D.J.: The state- based testing of object-oriented programs, Proc. IEEE Conf. Software Maintenance, 1993, str. 302-310.
[UML] http://www.uml.org/ (2014)
[Voa97] Voas J.: Fault Injection for the Masses, *Computer*, vol. 30, str. 129–130, 1997
[Voa03] Voas J.: Assuring Software Quality Assurance, *IEEE Software*, vol. 20, no 3, 2003
[Wag08] Wagner S., Deissenboeck F., Winter S.: Managing quality requirements using activity-based quality models, w Proc. 6th Int. Workshop on Software Quality (WoSQ '08), str. 29–34, ACM, 2008
[Wag09] Wagner S., Mendez Fernandez D., Islam S., Lochmann K.: A Security Requirements Approach for Web Systems, w Proc. Workshop Quality Assessment in Web (QAW 2009), 2009
[WAR] http://www.inf.fu-berlin.de/lehre/SS03/19560-P/Docs/JWSDP/tutorial/doc/WebApp3.html (dostęp XI 2014)
[WBB13] Winkler D., Biffl S., Bergsmann J. (Eds.): Software Quality Increasing Value in Software and Systems Development, 5th Int. Conf., SWQD 2013, LNBIP vol. 133, Springer,2013
[Win08] Winter S., Wagner S., Deissenboeck F.: A Comprehensive Model of Usability, w Engineering Interactive Systems, LNCS, vol. 4940, str. 106–122, Springer, 2008.
[weka] weka: http://www.cs.waikato.ac.nz/ml/wek
[Wis09] Wiszniewski B., Bereza-Jarociński B.: Teoria i praktyka testowania programów, PWN 2009
[Wit05] Witten I.H., Frank E.: Data Mining: Practical machine learning tools and techniques, 2nd ed. Morgan Kaufmann, San Francisco, 2005

[Woo06] Woodward M.R., Hennell M.A.: On the relationship between two control-flow coverage criteria: all JJ-paths and MCDC, *Inf. & Soft. Technology*, vol. 48, str. 433-440, 2006
[Zha08] Zhang L., Lin L., Gao H.: 2-D Software quality model and a case study in software flexibility research, CIMCA 2008, ss. 1147-1152, DOI: 10.1109/CIMCA.2008.70

4. Omówienie pozostałych osiągnięć naukowo-badawczych

Prace badawcze prowadzone przeze mnie po doktoracie dotyczą szeroko pojmowanej inżynierii oprogramowania ukierunkowanej na wyprodukowanie oprogramowania wysokiej jakości oraz poprawy jakości już istniejącego oprogramowania.

Jeden z nurtów mojej działalności naukowej dotyczy szeroko pojętego modelowania. Początkowo zajmowałam się wykorzystaniem sieci Petri do modelowania i badania wydajności systemów produkcyjnych a w ostatnich latach także systemów komponentowych.

Interesowałam się zastosowaniem komponentów w budowie oprogramowania. Pod moim kierownictwem powstało w Instytucie Informatyki PW w 2006 roku repozytorium przechowujące i wyszukujące komponenty, które daje lepsze efekty wyszukiwania produktów oraz dostawców niż istniejące ówcześnie repozytoria. Było to możliwe dzięki zastosowaniu zaproponowanego zbioru cech komponentowych do opisu komponentów. Problematyce modeli komponentowych poświęcone jest kilka prac napisanych przy współudziale moich dyplomantów, których udział oceniam na 20-30% .

Zajmowałam się także wytarzaniem oprogramowania z wykorzystaniem modeli (ang. *MDE - Model Driven Engineering)* w szczególności w notacji UML (ang. *Unified Modeling Language*) oraz innych specjalizowanych modeli. Prowadziłam także prace dotyczące transformacji modeli do UML, badaniem spójności tych modeli. Tematyce tej poświęcone jest kilka publikacji napisanych wspólnie z moimi dyplomantami (studia inżynierskie i magisterskie). Mój udział w tych badaniach oceniam na 50-70%.

Inny nurt moich badań dotyczy metryk oprogramowania i ich wpływu na testowalność, jakość i ewolucję oprogramowania. Pomiary tych metryk pozwalają uzyskać informację o słabych punktach projektu. Wyniki tych prac są przedstawione w 14 pracach (rozdziały w książce w języku angielskim, polskim, recenzowane konferencje międzynarodowe).

Kolejny nurt działalności naukowej dotyczy różnych metod testowania oprogramowania i narzędzi wspomagających, automatyzujących testowanie w szczególności generacji testów z modeli UML. Pod moim kierunkiem powstały narzędzia wspomagające i automatyzujące proces testowania oprogramowania. Powstało wiele prac – rozdziały w książkach, recenzowane konferencje międzynarodowe. Prowadziłam także prace porównujące efektywność testowania mutacyjnego i opartego o przepływ danych. Kolejne badania dotyczyły ograniczania zbioru mutantów przy zachowaniu akceptowalnej jakości testowania (np. wysokiego pokrycia kodu).

Interesowałam się także integracją systemów softwarowych. Tematyki tej dotyczą rozdziały w książkach w języku angielskim (1) oraz polskim (2) napisane przy współudziale mojego dyplomanta W. Kiermasza.

Prowadziłam także prace dotyczące architektury zorientowanej na usługi (ang. SOA service oriented architecture), wyniki przedstawione zostały w 9 publikacjach. Pod moim kierunkiem został opracowany w 2006 roku w Instytucie Informatyki monitor usług działający na poziomie szyny usług (ang. Enterprise Service Bus) i przygotowany do zastosowania w środowisku integracyjnym webMethods. Istniejące w owym czasie narzędzia pozwalały na monitorowanie na poziomie procesów biznesowych i nie pozwalały na monitorowanie usług na poziomie szyny usług w realizacjach rozproszonych. Wyniki eksperymentów przeprowadzonych na rzeczywistej architekturze SOA przedstawiłam, wspólnie z moim studentem M. Wardą, w 2 pracach. Prowadziłam także badania dotyczące wykrywania anomalii w systemach SOA za pomocą algorytmów wydobywania wiedzy. Do badań tych powstał specjalny system gdyż tego typu badań nie można było przeprowadzać na

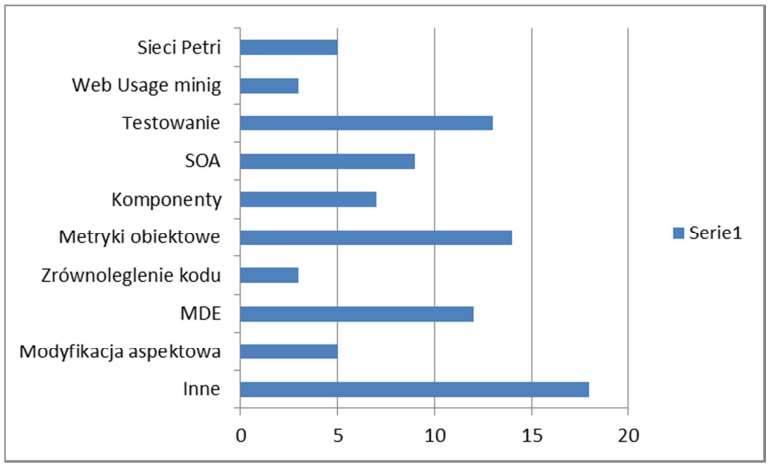
rzeczywistych systemach. Wyniki prac zaprezentowane zostały na konferencjach międzynarodowych

Zajmowałam się zagadnieniami związanymi ze zrównoleglaniem kodu w języku C (3 prace). Po moim kierunkiem zbudowane zostało specjalne środowisko do zrównoleglania i wizualizacji kodu w C oraz wtyczka do eclipsa. Kierowałam szeregiem eksperymentów realizowanych przez moją dyplomantkę J. Fugas, w zrealizowanym środowisku. Wyniki badania efektywności automatycznego i manualnego zrównoleglania kodu wykonywanego na maszynie 2 i 16 procesorowej opisane są w pracy zaprezentowanej na konferencji międzynarodowej w 2010 roku.

Kierowałam projektowaniem i realizacją systemu typu *web usage mining*. Spośród innych systemów typu *web usage mining* system ten wyróżnia możliwość dołączania go do różnych aplikacji internetowych oraz dostosowywania do potrzeb. Stanowi on narzędzie zarówno rejestrujące jak i analizujące dane za pomocą algorytmów do odkrywania wiedzy. System udostępnia algorytmy należące do czterech głównych kategorii metod odkrywania wiedzy: klasyfikacji, odkrywania reguł asocjacyjnych, grupowania oraz wykrywania wzorców sekwencyjnych, można w nim także określać jakie dane mają być rejestrowane. Wyniki szeregu eksperymentów przeprowadzonych pod moim kierunkiem przez A. Orlewicz (dyplom magisterski) opublikowano jako rozdziały w książkach w języku angielskim.

W latach 2007- 2009 prowadziłam także prace mające na celu modyfikacje funkcji oprogramowania napisanego w Javie bez dostępu do jego kodu źródłowego za pomocą programowania aspektowego. Tematyce tej poświęcone jest 5 prac napisanych wspólnie z K. Billewiczem moim dyplomantem.

Liczbę opublikowanych prac w poszczególnych dziedzinach inżynierii oprogramowania pokazano na Rys. 1. Większość prac powstała przy współudziale moich dyplomantów (dyplomy magisterskie i inżynierskie). Mój udział w tych publikacjach oceniam na co najmniej 70 %.



Rys. 1 Opublikowane prace w wybranych działach będących przedmiotem badań

4.1 Osiągnięcia naukowo badawcze – analiza bibliometryczna

Osiągnięcia naukowe po uzyskaniu stopnia doktora przedstawiłam sumarycznie w Tabeli 1.

Tabela 1 Charakterystyka liczbowa dorobku naukowego **po doktoracie**

Rodzaj	
Rozdziały w książkach w języku angielskim	22
Rozdziały w książkach w języku polskim	12
Artykuły w czasopismach w języku angielskim	12
Artykuły w czasopismach w języku polskim	12
Artykuły w materiałach konferencji międzynarodowych	19
Artykuły w materiałach konferencji krajowych	4
Książki/skrypty w języku polskim	1
Raporty Instytutu Informatyki PW	4
Razem	86

Przed doktoratem miałam tylko 2 publikacje.

Cytowania

Według *Google Scholar* w lutym 2015 do moich prac były **102 cytowania**, a 76 bez autocytowań, **h-index=5**. Od 2010 – łączna liczba cytowań to 51 a h-index=3.

W bazie *Web of Science* (WoS) znajduje się 17 moich prac z lat 2000-2015 i dla nich h-index=1.

W bazie *Scopus* są 23 moje prace do których jest 21 cytowań a obliczony h-index=2.

W repozytorium Politechniki Warszawskiej:
http://www.ii.pw.edu.pl/ii_pol/Instytut-Informatyki/Dzialalnosc-naukowa/Repozytorium-Instytutu/Publikacje-pracownikow

jest wpisanych 75 pozycji dających łącznie **316 punktów** (wg. punktacji MNiSW).

Ilona Bluemke