Zarządzanie Przedsiębiorstwem. Enterprise Management
Volume 22 • Number 4 • December 2019 • pp. 22–28
DOI: 10.25961/ent.manag.22.04.04

MODEL I METODA OCENY JAKOŚCI IMPLEMENTACJI WZORCÓW PROJEKTOWYCH

Rafał Wojszczyk

Wydział Elektroniki i Informatyki, Politechnika Koszalińska, Poland

Corresponding author:
Rafał Wojszczyk
Politechnika Koszalińska
Wydział Elektroniki i Informatyki
Śniadeckich 2, 75-453 Koszalin, Poland
phone: (+48) 94 34 78 706
e-mail: rafal.wojszczyk@tu.koszalin.pl

THE MODEL AND THE METHOD OF QUALITY ASSESSMENT OF DESIGN PATTERN IMPLEMENTATION

Abstract
Design patterns are very popular in programmers. It allow you to solve choosen problems in the object-oriented languages, and also provide some benefits, for example: lower cost of program upgrades. Implementations of patterns (even the same kind) can vary considerably from one another and do not provide the expected benefits. The paper presents a solution for measuring the implementation of patterns, and then provides a verification of this solution in practice and on the space of a joint formal representation.

Keywords
Design patterns, software quality, agile methodologies for software development.

1. Wprowadzenie

Jednym z podstawowych wyzwań, z jakim spotykają się producenci oprogramowania jest konieczność naprawiania błędów i rozwoju kodu programu. Wynika to m.in. z konieczności dostosowania wytwarzanego oprogramowania do nowych regulacji prawnych oraz z konkurencyjności na rynku. Wytwórca musi sprostać temu wyzwaniu, ponieważ zobowiązuje się do tego w ramach gwarancji udzielanej na sprzedawany produkt, za co klient nie powinien ponosić dodatkowych kosztów – zatem koszty ponosi wyłącznie wytwórca. Programiści stosują różne rozwiązania, które pomagają w obniżeniu kosztów rozbudowy i modyfikacji oprogramowania. W niniejszej pracy, skupiono się na jednym z takich rozwiązań, które łączy paradygmat programowania obiektowego i dobre praktyki wynikające z doświadczenia wielu programistów – są to wzorce projektowe.

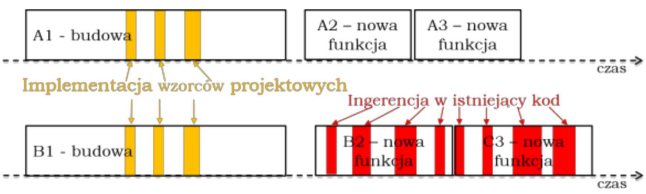
Wzorce projektowe opisane w [6] to *szkielety gotowych mechanizmów, które można wykorzystywać przy rozwiązywaniu typowych problemów projektowania i programowania*. Wzorce to pewnego rodzaju abstrakcja w programowaniu, którą w przypadku każdego wystąpienia wzorca należy skonkretyzować i dopasować do kodu programu, w którym aplikowany jest wzorzec. Skonkretyzowane wystąpienie wzorca i dopasowane do kodu programu stanowi implementację tego wzorca.

Drugi rozdział niniejszej pracy przedstawia kontekst zagadnienia i sformułowanie problemu. W trzecim rozdziale zaprezentowano tytułowy model oraz metodę opartą na tym modelu, natomiast czwarty rozdział zawiera badania weryfikacyjne. Piąty rozdział stanowi podsumowanie pracy.

2. Kontekst zagadnienia

2.1. Jakość implementacji wzorców projektowych

Przykładem popularnego wzorca jest strategia (ang. *strategy*) [4], celem tego wzorca jest za [6]: zgrupowanie rodziny algorytmów (np. różne rodzaje sortowania) i ich zamienne wykorzystywanie. W kontekście niskich kosztów rozbudowy oznacza to, że dodając nowy algorytm (tzn. rozbudowując program o nową funkcję), zgodnie z tym wzorcem, programista powinien poświęcić czas na dodanie nowego algorytmu, bez ingerowania w istniejący kod. Każda ingerencja wiąże się do dodatkową pracą programistów i testerów. Taką sytuację, naniesioną na oś czasu, przedstawia górny fragment rys. 1.



Rys. 1. Górny fragment przedstawia oś czasu z oczekiwaną budową oprogramowania zawierającego wzorce projektowe, dolny fragment przedstawia oś czasu z ingerencją w istniejący kod.

Ramka oznaczona jako „A1 – budowa" na rys. 1 oznacza czas poświęcony na zbudowanie bazowej funkcjonalności oprogramowania, w tym czas poświęcony na implementację wzorców. Natomiast ramki A2 i A3 to dodanie nowych funkcji, np. nowych algorytmów we wzorcu strategia.

Górny fragment rys. 1 przedstawia sytuację, która jest na ogół pożądanym zjawiskiem, w praktyce jednak bywa, że występuje sytuacja, którą przedstawia dolny fragment rys. 1. Wówczas ramka B1 oznacza czas poświęcony na budowę bazowej części oprogramowania i implementację wzorców. Ramki B2 i B3 to również rozbudowa oprogramowania, jednakże w tym przypadku występują dodatkowe koszty związane z ingerencją w istniejący kod programu. Jest to spowodowane przez usterki w implementacji wzorców, popełnione jeszcze w trakcie bazowej implementacji, tzn. B1.

Wzorce za każdym razem są implementowane przez człowieka, ponieważ wymagają wybrania odpowiedniego wariantu i indywidualnego dostosowania do kodu programu. Z tego powodu każde wystąpienie danego wzorca będzie inne (nawet w jednym programie komputerowym) pomimo, że to będzie nadal ten sam wzorzec. Aktualnie nie ma jednego powszechnie przyjętego standardu implementacji wzorców, ani też żadna organizacja nie sprawuje nadzoru nad wzorcami. Katalog [6] zawiera wyłącznie przykłady implementacji. Znaczna różnorodność w implementacji wzorców to jedna z przyczyn powstawania usterek w implementacji wzorców. Często też programiści przyczyniają się do powstawania usterek, poprzez implementację wzorców, których nie rozumieją lub nieświadomie popełniając błędy [26]. Kolejna przyczyna usterek to brak szczegółowego opisu wzorców w dokumentacji projektowej, dotyczy to przede wszystkim zespołów programistycznych pracujących według metodyk zwinnych (ang. *agile*), gdzie zgodnie z zasadami zwinności dokumentacja projektowa ograniczona jest do minimum. Usterki w implementacji wzorców projektowych świadczą o niskiej jakość tej implementacji.

Jakość w ogólnym ujęciu definiowana jest w podobnych słowach: „*właściwe wykonanie*" [17], „*dobroć produktu*" [10] oraz „*stopień, w jakim moduł, system lub proces spełnia określone wymagania, potrzeby lub oczekiwania*" [9] za [17] – warto zauważyć, że wymienione definicje łączy wspólna cecha, tj. produkt o wysokiej jakości zapewnia pewne korzyści (dobroć, spełnia oczekiwania). Implementacja danego wzorca projektowego w oprogramowaniu może zapewniać korzyść, np. niski koszt rozbudowy oprogramowania. Na potrzeby oceny jakości implementacji wzorców projektowych przyjęto skalę oceny od 0 do 3, co przedstawia rys. 2.

Powszechnie przyjęło się, że programiści implementują wzorce na 2 stopniu jakości, tj. tak, aby implementacja spełniała wyłącznie postawiony cel. Niepożądany jest 1 stopień jakości implementacji, taka implementacja nie spełnia celu i zawiera błędy. Zarówno 1 i 2 stopień jakości implementacji nie zapewniają korzyści, tzn. implementacja wzorca nie przyczynia się do obniżenia kosztów rozbudowy. Zapewnia to dopiero implementacja na 3 stopniu jakości.

Rys. 2. Stopnie jakości implementacji wzorców projektowych.

2.2. Alternatywne metody

Jakość kodu źródłowego powszechnie jest kojarzona z metrykami oprogramowania obiektowego i modelami jakości, jednakże tego typu metody nie mają zastosowania do implementacji wzorców projektowych, ponieważ są za mało dokładne. Z drugiej strony koszt użycia popularnych metryk jest dość niski i dopuszczalny w zwinnych zespołach (tj. zespół programistów pracujących według metodyki zwinnej, np. Scrum [26]). Dokładność metod, które analizują implementację wzorców projektowych, to w uproszczeniu liczba analizowanych elementów we wzorcu (każdy wzorzec można zdekomponować do pojedynczych elementów, tzw. własności wzorca) w danym kryterium oceny. Natomiast na koszt użycia składa się wiele czynników i trudno jest wskazać jedną miarę kosztu użycia.

Kolejna grupa alternatywnych metod to badania naukowe związane z analizowaniem implementacji wzorców [1], gdzie dominuje problem wyszukiwania wystąpień wzorców projektowych. Wynikiem działania metody wyszukującej wystąpienia jest liczba egzemplarzy wzorców w badanym fragmencie kodu programu lub odpowiedniku kodu. Większość takich metod wyszukujących wystąpienia wzorców działa binarnie, tj. wskazuje wystąpienie wzorca lub brak wzorca, co odpowiada ocenie jako 2 lub 0 stopień jakości implementacji, jest to zbyt mała dokładność. Koszt użycia metod wyszukujących wystąpienia wzorców jest w większości przypadków akceptowalny przez zwinne zespoły wytwórcze. Inne badania dotyczą metod weryfikacji implementacji wzorców, których wynikiem jest wskazanie fragmentu kodu, który jest zgodny z szablonem wzorca. Pełna zgodność z szablonem odpowiada 2 stopniu jakości implementacji, natomiast odstępstwa od tego odpowiadają 1 i 0 stopniu. Koszt użycia metod weryfikujących implementację wzorców jest większy niż możliwości zwinnego zespołu, ponieważ wymagana jest szczegółowa dokumentacja. Tabela 1 przedstawia zbiorcze zestawienie wymienionych badań oraz odpowiadający im oceniany stopień jakości implementacji wzorców. Reasumując, alternatywne metody nie pozwalają na odróżnienie im-

Tabela 1
Zbiorcze zestawienie alternatywnych metod.

Rodzaj	Odpowiadające stopnie jakości	Koszt	Przykłady metod
Klasyczne modele i metryki oprogramowania	0, 1	Dopuszczalny	FRUPS [9], CUPRIMDA [9, 10, 23]
Wyszukiwanie wystąpień wzorców projektowych	0, 1, 2	Dopuszczalny	[1, 4, 6, 11, 14, 16, 17, 18]
Weryfikacja implementacji wzorców projektowych	0, 1, 2	Większy niż możliwości zwinnego zespołu	[2, 12, 13]

plementacji zgodnej z 3 stopniem od 2 stopnia jakości (tj. nie można ocenić czy implementacja danego wzorca zapewnia oczekiwane korzyści) oraz nie mogą być stosowane przez zwinne zespoły.

2.3. Sformułkowanie problemu

W nawiązaniu do poprzedniej sekcji, można stwierdzić, że istnieje potrzeba opracowania nowej metody oceny jakości implementacji wzorców projektowych. Metoda powinna być przede wszystkim przeznaczona do wzorców projektowych oraz nie powinna zakłócać zasad panujących w zwinnych zespołach wytwórczych, aby nie zabierać zbyt wiele czasu z codziennej pracy, w uzyskaniu odpowiedzi na pytanie *„czy dana implementacja wzorca spełnia wymaganą jakość?"*. Przedstawione pytanie determinuje konieczność podstawienia poniższych założeń i ograniczeń.

Dany jest zbiór szablonów wzorców projektowych. Jednocześnie dany jest bazodanowy program komputerowy, w postaci kodu źródłowego lub pośredniego, dokładniej kod warstwy logiki biznesowej, dostępu do danych, tzw. back-end. Program jest stale rozbudowywany. W kodzie programu występują implementacje wzorców projektowych, dla każdego wystąpienia wzorca znane jest miejsce wystąpienia (tzw. rdzeń wzorca), wariant oraz kryterium oceny. Dany jest również obszar dopuszczalnych wyników metryk dla tego kryterium oceny. Problem badawczy z punktu widzenia lidera zespołu programistów sprowadza się do odpowiedzi na pytanie:

Czy istnieje metoda oceny jakości implementacji wzorców projektowych, która pozwala uzyskać wynik z lepszą dokładnością i nie większym kosztem niż jest to możliwe przy pomocy alternatywnych metod?

W związku z niepowodzeniem poszukiwań takiej metody, podjęto badania ukierunkowane na zbudowanie nowego, oryginalnego rozwiązania. Przedsięwzięcie to wymagało opracowania wcześniej formalnego modelu, na bazie którego zbudowano metodę.

3. Model i metoda oceny jakości implementacji wzorców projektowych

3.1. Model formalny

Model proponowany w niniejszej pracy rozumiany jest jako zbiór danych odwzorowujących wybrany obszar świata rzeczywistego, ograniczenia narzucone na dane oraz związki występujące pomiędzy danymi.

Pierwszy z modelowanych obszarów odpowiada programowi komputerowemu (odwzorowuje kod źródłowy w skończony zbiór danych), dlatego został nazwany **Modelem Programu (MP)** i odwzorowuje paradygmat obiektowy. Formalnie model programu jest zbiorem danych opisujących kod programu komputerowego, ograniczeń wynikających z paradygmatu obiektowego oraz związków pomiędzy danymi.

Drugi obszar to **Model Wzorca (MW)**, który odzwierciedla wzorce projektowe. Występowanie tego obszaru determinuje konieczność formalizacji szablonów wzorców projektowych, tzn. konieczne jest przedstawienie charakterystyk (inaczej cech), z których składa się dany wzorzec. Formalnie model wzorca to zbiór danych, które opisują charakterystyki oraz związki między nimi.

Trzeci obszar to **Model Metryk (MM)**, jest niezbędny, aby szczegółowo mierzyć implementację wzorców oraz wyrazić jakość w sposób liczbowy, tj. zgodnie ze schematem jakości technicznej [13], gdzie metryki uszczegóławiają charakterystyki. Każda metryka jest wyrażona przez z jedną z funkcji oraz zbiór danych wskazujący na sposób interpretacji wyniku funkcji, zgodnie z przyjętym kryterium oceny. Formalnie model metryk to zbiór danych, które pozwalają odwzorować szablonowe elementy programowania obiektowego, zbiór funkcji matematycznych, które odwzorowują metryki oprogramowania oraz zbiór zależności opisujących powiązania pomiędzy wspomnianymi zbiorami danych i funkcji.

Wykorzystując wymienione wyżej obszary opracowano model, który formalnie wyrażony jest krotką:

$$(\mathrm{MP}, (\mathrm{MW}, \mathrm{MM})), \tag{1}$$

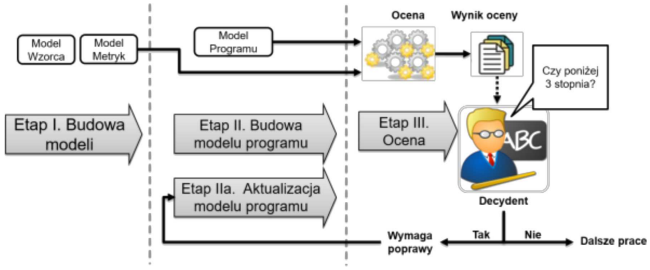
gdzie MP to model programu, MW to model wzorca, MM to model metryk. Model programu budowany jest na podstawie kodu źródłowego lub ekwiwalentu tego kodu, co stanowi jeden z parametrów wejściowych. Pozostałe parametry wejściowe to para model wzorca i model metryk, które budowane są na podstawie wiedzy i doświadczenia (tzn. dobrych praktyk i metryk oprogramowania), a następnie wykorzystywane jako szablon wzorca.

3.2. Zasada działania metody

Proponowana metoda to uporządkowane postępowanie, które korzystając z elementów opisanego w po-

przedniej sekcji modelu, pozwala ocenić jakość implementacji wzorców projektowych.

W metodzie wyróżniono 4 etapy, co przedstawia rys. 3 [24] (4 etap nie został zaznaczony). W procedurze postępowania przewidziano, że pierwszy etap wykonywany jest jednorazowo wraz z wdrożeniem metody do zespołu, a następne dwa etapy wykonywane cyklicznie, przynajmniej raz w każdej iteracji (w metodykach zwinnych). Etapy I, II oraz III winny być wykonywane obligatoryjnie, natomiast IIa jest warunkowy i zależy od potrzeb zespołu.



Rys. 3. Schemat użycia metody.

Etap I – budowa modelu w obszarze wzorców i metryk – występuje jednonarodowo, ponieważ w kolejnych etapach modele będą wyłącznie wykorzystywane, lub w przypadku etapu IV rozbudowywane. Odpowiedzialność za budowę modelu ponosi w dużej mierze lider zespołu, ponieważ najczęściej to on posiada największe doświadczenie, jak również wiedzę o wzorcach projektowych.

Etap II – budowa modelu w obszarze programu – występuje cyklicznie, przynajmniej jeden raz w trakcie iteracji metodyk zwinnych, każdorazowo przed etapem III. Zalecana jest automatyzacja tego etapu [22], ponieważ polega to na przekształceniu kodu programu lub ekwiwalentu kodu do modelu programu. Etap aktualizacji modelu programu, zaznaczony na rys. 3 jako IIa, występuje zamiennie zamiast etapu II, gdy zachodzi potrzeba ponownej oceny.

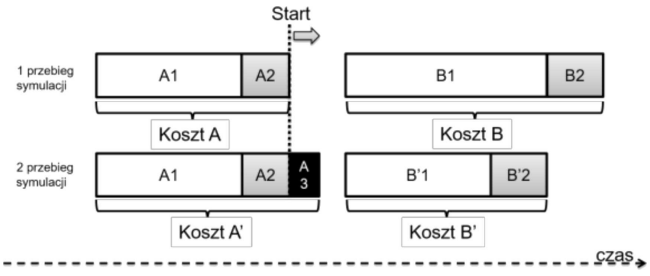
Etap III – ocena – występowanie etapu III w cyklu wytwórczym oprogramowania zależy od preferencji zespołu wytwórczego. Zalecany czasokres to czas zarezerwowany na inspekcje oraz poprawę kodu. Etap oceny może być stosowany częściej niż jednorazowo w trakcie iteracji, jednakże powinien występować po wykonaniu potencjalnie gotowego fragmentu kodu źródłowego. Zalecaną osobą odpowiedzialną za wykonanie oceny jest lider zespołu, ponieważ często dysponuje największym doświadczeniem. Bazując na nim podejmuje decyzje, czy jakość implementacji wzorców projektowych (wytworzona przez zespół, za który odpowiada) jest wystarczająca w przyjętym kryterium oceny lub wymaga poprawy. W przypadku wymaganej poprawy implementacji wzorców projektowych należy uzyskać podpowiedzi z modelu metryk, które wskażą programistom fragmenty wymagające zmian oraz zalecany rodzaj zmian. Lider zespołu podejmuje decyzje opierając się na wyniku oceny. Wynik oceny to stopień jakości implementacji, wyrażony dla każdego elementu wzorca projektowego.

4. Badania weryfikacyjne

4.1. Weryfikacja w środowisku akademickim

Wstępną weryfikację metody przeprowadzono w środowisku akademickim. Eksperyment polegał na zasymulowaniu fragmentu iteracji procesu wytwórczego [23], w której rozbudowywane jest oprogramowanie zawierające wzorzec Strategia. Zasadniczym celem tego eksperymentu było wykazanie, że czas rozbudowy oprogramowania (poprzez dodanie nowej operacji do wzorca Strategia) będzie krótszy w przypadku implementacji wzorca zgodnie z wytycznymi wynikającymi z metody, względem czasu potrzebnego na rozbudowę, gdy implementacja wzorca zawiera usterki. W eksperymencie wzięli udział trzej studenci III i IV roku kierunku Informatyka. Niestety, w eksperymencie nie możliwe było odwzorowanie środowiska typowego jak w firmie programistycznej, więc konieczne było wiele uproszczeń, dotyczyło to m.in.: wykorzystanego kodu źródłowego, symulacji wyłącznie fragmentu iteracji związanego z rozbudową.

Symulacja składała się z dwóch przebiegów, które przedstawia rys. 4. W pierwszym przebiegu studenci brali udział wyłącznie w iteracji B. Iteracja A zawierała wykonanie oprogramowania wraz z kilkoma usterkami w implementacji wzorca strategia, przy czym studenci nie byli obecni. Przystępując do pracy z kodem dopiero w iteracji B, otrzymali tzw. kod zastany. W trakcie iteracji B studenci mieli za zadanie dodać nową operację do strategii oraz naprawić zauważone wcześniej błędy. Takie samo zadanie powierzono w iteracji B', jednakże w przypadku drugiego przebiegu symulacji studenci brali udział również w A3. Iteracja A1 w drugim przebiegu symulacji jest taka sama jak w pierwszym przebiegu, różnice występują w czasie przeznaczonym na inspekcje kodu (A2), gdzie dodatkowo wykorzystano proponowaną metodę oraz poprawiono implementację zgodnie z metodą – co również wykonali studenci. Następnie przystąpili do iteracji B', która ponownie przewidywała rozbudowę oprogramowania poprzez dodanie nowej operacji do strategii oraz naprawę zauważonych wcześniej błędów, jednakże w tym przypadku implementacja wzorca Strategia nie zawierała usterek jak w przypadku iteracji B.



Rys. 4. Schemat symulacji.

Porównanie poniesionych nakładów pracy przez każdego z uczestników przedstawia tabela 2. Po przeanalizowaniu tabeli wynika, że wzbogacając proces wytwórczy o proponowaną metodę, a następnie poprawę jakości (jeśli zachodzi taka potrzeba), to można uzyskać zysk czasowy, pomimo dodatkowego czasu poświęconego w iteracji A3. Pracochłonność wykonania oceny jest pomijalna, ponieważ może to być w znacznym stopniu zautomatyzowane [22].

Tabela 2
Wyniki eksperymentu akademickiego.

Uczestnik	Czas wykonania zadań w B [rbh]	Czas wykonania zadań w B' [rbh]	Zysk po zastosowaniu metody
Student nr 1	3,5	2	43%
Student nr 2	5,5	4	28%
Student nr 3	7	6	14%

Reasumując, przedstawione wyniki, w perspektywie kosztów rozbudowy, pokazują zysk czasowy średnio 28% przy zastosowaniu metody. Wynik ten potwierdza, że proponowana metoda stanowi rozwiązanie przedstawionego wcześniej problemu i umożliwia zapewnienie korzyści związanych z implementacją wzorców projektowych. Warto dodać, że w przedstawionych wynikach nie uwzględnia się kosztów budowy modeli i wdrożenia takiej metody, ponieważ jest to koszt jednorazowy.

4.2. Eksperyment weryfikacyjny w środowisku komercyjnym

Weryfikację metody przeprowadzono przy współpracy z firmą Quick-Solution z Koszalina, która udostępniła kod źródłowy. Kod składa się z 11 projektów (pakietów), 52 unikalnych klas, które zawierają łącznie 3374 wiersze kodu.

Wymagania stawiane udostępnionemu systemowi to możliwość rozbudowy o nowe rozwiązania oraz możliwość szybkiego naprawienia błędów, które nie spowodują lawinowego powstania innych błędów. Wzorce projektowe zaimplementowano jako realizację wymienionych wymagań. Zatem oczekuje się, że wybrane fragmenty kodu (implementacje wzorców) będą umożliwiały rozbudowę oprogramowania niższym kosztem po użyciu metody, niż bez niej. Cel eksperymentu to znalezienie odpowiedzi na pytanie: czy proponowana metoda pozwala obniżyć koszty rozbudowy oprogramowania? A jeżeli tak, to ile szacunkowo wynosi oszczędność? Zmniejszenie kosztu rozbudowy oprogramowania możliwe jest poprzez wczesne wykrycie i poprawę usterek w implementacji wzorców projektowych.

W ramach prac przygotowawczych ustalono rodzaje wzorców oraz sposób współpracy. Prace związane bezpośrednio z firmą rozpoczęto od opracowania szablonów wzorców (co odpowiada pierwszemu etapowi proponowanej metody), były to wzorce Polecenie i Fabryka. Następnie w drugim etapie eksperymentu wykonano ocenę jakości implementacji wzorców projektowych. W ostatnim etapie zweryfikowano i omówiono wyniki oceny. W wyniku oceny implementacji wzorców znaleziono 7 usterek, tzn. 7 fragmentów kodu nie spełniało 3 stopnia jakości, były to m.in.:

- typ Command w wzorcu polecenie był klasą zamiast interfejsem,
- nie wszystkie polecenia konkretne były uwzględnione w mapie połączeń wzorca Polecenie, należało dodać brakujące polecenia do mapy,
- modyfikator wewnętrzny (ograniczony zasięg dostępności metody) w głównej metodzie wzorca fabryka, należy zmienić modyfikator dostępu na publiczny.

Koszt zmian do wykonania wynikających z wykrytych usterek we wzorcu Fabryka jest niewielki i można tą usterkę poprawić w trakcie innych prac. Znacznie więcej zainteresowania wzbudziły wyniki oceny jakości wzorca Polecenie. W opinii zespołu Quick-Sotulion „*zlokalizowanie tych błędów na początku fazy implementacji jest niezwykle efektywne i zaoszczędza znacznie więcej czasu niż w przypadku, gdy kod zostałby wydany w wersji produkcyjnej*". Koszt prac bez użycia metody zespół Quick-Solution oszacował na 20 rbh, co zawierało czas poświęcony na szukanie usterek, testowanie, prace implementacyjne. Do szacowania wykorzystano mieszaną metodę, bazującą na ocenie eksperta oraz szacowania przez analogię. Po zapoznaniu się z wynikami oceny jakości zespół oszacował koszt prac na 6 rbh (prace implementacyjne), zatem szacunkowa oszczędność w rozbudowie oprogramowania wynosi 14 rbh. Przedstawiona czasochłonność nie uwzględnia czasu potrzebnego na opracowanie referencyjnych implementacji, jednakże jest to koszt jednorazowy. Raz opracowane modele będą wykorzystywane wielokrotnie w produkcyjnym zastosowaniu metody.

4.3. Analiza porównawcza dokładności metody

Oprócz praktycznych eksperymentów przeprowadzono również dwie analizy porównawcze. Niestety, nie możliwe jest bezpośrednie porównanie alternatywnych metod pod względem dokładności, tj. na zasadzie benchmarku, wynika to z różnego przeznaczenia metod oraz zastosowania do odmiennych języków programowania [16]. Większość z alternatywnych metod przeznaczona jest do wykrywania wystąpień wzorców projektowych, dodatkowo te metody ograniczone są do najpopularniejszych implementacji wzorców zapewniających jedynie cel, czyli 2 stopień jakości implementacji. Bezpośrednie porównanie proponowanej metody z metodami wyszukiwania wystąpień jest niemiarodajne, ponieważ wynik metod wyszukujących (liczba wystąpień danego wzorca) nie zawiera informacji o jakości implementacji tych wystąpień.

Po uwzględnieniu wymienionych trudności w przeprowadzeniu bezpośredniego porównania, wybrano metody o podobnym przeznaczeniu: [3, 14, 15]. Następnie na przestrzeni wspólnej reprezentacji wykonano analizę porównawczą tychże metod, której celem było wykazanie większej dokładności w analizowanych własnościach wzorców projektowych (co jest niezbędne do od-

różnienia 2 i 3 poziomu jakości implementacji). Analizę porównawczą przeprowadzono dekomponując własności wzorców projektowych, które są analizowane przez metody. Ograniczono się do wzorców Singleton i Strategia, które stanowią wyczerpujący przykład. Każdej własności [25], w podziale na porównywane metody, przypisano odpowiednią wartość punktową:

- 0 – metoda uniemożliwia pomiar danej własności wzorca,
- 0,5 – metoda mierzy własność niedokładnie, lub nie uwzględnia wszystkich elementów w danej własności,
- 0,7 – specyfikacja metody umożliwia zmierzenie danej własności, ale autor danej metody nie uwzględnił tego w zastosowaniu do danego wzorca,
- 1 – metoda mierzy daną własność bez zastrzeżeń.

W tabeli 3 przedstawiono sumaryczne zestawienie wyników. Po wnikliwej analizie wyników cząstkowych zauważono, że dokładność w alternatywnych metodach jest zaniżana przez własności drobnoziarniste, tj. występujące na poziomie pojedynczych wierszy kodu. Następnym czynnikiem zmniejszającym dokładność alternatywnych metod jest ograniczona liczba modyfikatorów ogólnych i modyfikatorów dostępu. W sytuacji, gdy oczekiwany jest dokładnie jeden z modyfikatorów jest to oczywiste. Natomiast w pozostałych przypadkach, gdy inne modyfikatory są dopuszczalne ogranicza to dokładność.

Tabela 3
Sumaryczne zestawienie dokładności.

Wzorzec/metoda	Proponowana metoda	[3]	[15]	[14]
Singleton	25	19	11	11
Strategia	25	17	9	8

4.4. Analiza porównawcza kosztu użycia metody

Przy wyborze metod do porównania pod względem kosztu użycia ograniczono się do metod wybranych w poprzedniej sekcji, z wyłączeniem metody [14] ze względu na brak dostatecznych informacji o kosztach użycia tej metody. Koszt użycia metody jest ważnym czynnikiem w zwinnych metodykach wytwórczych.

Koszt użycia metody można podzielić na dwa rodzaje: koszty jednorazowe ponoszone wstępnie, przed pierwszym użyciem metody oraz koszty powtarzające się przy każdym użyciu metody. Koszty jednorazowe to budowa szablonów wzorców projektowych, którą należy poprzedzić przyswojeniem odpowiedniej reprezentacji formalnej. W przeprowadzonym porównaniu ograniczono się do pojedynczych kosztów, tzn. szablon jednego wzorca, jedno użycie metody. Do kosztów powtarzających się należy zaliczyć m.in.: pozyskanie oprogramowania lub przekształcenie kodu źródłowego do postaci formalnej, wykonanie procesu oceny jakości (bądź weryfikacji implementacji w przypadku alternatywnych metod), rozbudowa szablonu wzorca o nowy wariant.

W proponowanej metodzie oraz [3] występują oba z wymienionych rodzajów kosztów. Natomiast w [15] konieczne jest tworzenie za każdym razem odpowiedniej dokumentacji, co powoduje, że nie można uznać tego jako koszt jednorazowy. Jako jednostkę miary kosztu zaproponowano umowną jednostkę kosztu użycia 1us – jeden umowny Singleton, który odpowiada pracochłonności potrzebnej do zdefiniowania szablonu wzorca projektowego Singleton w danej reprezentacji formalnej. Suma w/w kosztów nie odzwierciedla produkcyjnego kosztu użycia, tzn. dodanie nowego wariantu wykonywane jest raz na kilka iteracji, w przeciwieństwie do oceny, która jest wykonywana cyklicznie w każdej iteracji. Produkcyjny koszt użycia metod obliczono poprzez symulację, która bazuje na informacjach otrzymanych od zespołu Quick-Solution. Wynik analizy przedstawia tabela 4.

Tabela 4
Wynik porównania kosztu użycia metod.

	Proponowana metoda	[3]	[15]
Suma pojedynczych kosztów	2,9	5,3	6,7
Suma kosztów z symulacji	10,7	27	27,5

Wysokie koszty użycia alternatywnych metod, które wynikły po przeprowadzeniu symulacji występują przede wszystkim w wielokrotnie wykonywanych pojedynczych czynnościach, takich jak pozyskanie kodu źródłowego, czy uzyskanie sugestii poprawy.

5. Podsumowanie

W pracy opisano model i metodę oceny jakości implementacji wzorców projektowych, oraz wyniki związane z weryfikacją tej metody. W wyniku przeprowadzonej analizy porównawczej zweryfikowano dokładność większą o 25–68 % względem alternatywnych metod. Przekłada się to na możliwość odróżnienia 2 i 3 stopnia jakości implementacji, co jest niezbędne w zapewnieniu korzyści wynikających z implementacji wzorców projektowych. Natomiast koszt użycia niższy o 45–57 % umożliwia wykorzystanie proponowanej metody również przez zwinne zespoły wytwórcze. Weryfikacja praktyczna potwierdziła, że użycie metody pozwala na zysk czasowy 28% w środowisku akademickim i 20% w środowisku komercyjnym.

Pożądanym kierunkiem rozwoju od strony użytkowej jest dalsza ewaluacja modelu, tzn. dodanie większej liczby wzorców projektowych [5] oraz wzorców innego rodzaju (np. wzorców architektury). Dodatkowo bardzo praktyczne jest zautomatyzowanie wielu etapów w proponowanej metodzie w postaci narzędzia dla programistów, które byłoby zintegrowane z popularnymi środowiskami programistycznymi. Inny kierunek rozwoju to próba dodania do aktualnego modelu elementów [7], które pozwolą na zastosowanie do oprogramowania wielowątkowego.

Literatura

[1] Al-Obeidallah M.G., Petridis M., Kapetanakis S., *A Survey on Design Pattern Detection Approaches*, International Journal of Software Engineering (IJSE), 7, 3, 41–59, 2016.

[2] Binun A., *High Accuracy Design Pattern Detection*, Dysertacja doktorska, Rheinischen Friedrich Wilhelms Universitat Bonn, 2012.

[3] Blewitt A., *HEDGEHOG: Automatic Verification of Design Patterns in Java*, Dysertacja doktorska, University of Edinburgh, 2006.

[4] Czyczyn-Egird D., Wojszczyk R., *Determining the Popularity of Design Patterns Used by Programmers Based on the Analysis of Questions and Answers on Stackoverflow.com Social Network*, Communications in Computer and Information Science, Springer, 608, 421–433, 2016.

[5] De Lucia A. i inni, *Design pattern recovery through visual language parsing and source code analysis*, Journal of Systems and Software Archive, Elsevier Science Inc, New York, 82, 7, 2009.

[6] Gamma E. i inni, *WZORCE PROJEKTOWE Elementy oprogramowania wielokrotnego użytku*, Helion, Gliwice, 2010.

[7] Giebas D., Wojszczyk R., *Multithreaded Application Model*, Advances in Intelligent Systems and Computing, Springer International Publishing, 1004, 93–103, 2020.

[8] Grzanek K., *Realizacja systemu wyszukiwania wystąpień wzorców projektowych w oprogramowaniu przy zastosowaniu metod analizy statycznej kodu źródłowego*, Dysertacja doktorska, Politechnika Częstochowska, Łódź, 2008.

[9] IEEE 610 Standard Glossary of Software Engineering Terminology, The Institute of Electrical and Electronics Engineers, Nowy Jork, 1990.

[10] Kan S.H., *Metryki i modele w inżynierii jakości oprogramowania*, PWN SA, Warszawa, 2006.

[11] Khaer Md.A. i inni, *An Empirical Analysis of Software Systems for Measurement of Design Quality Level Based on Design Patterns*, Computer and Information Technology, IEEE, 2007.

[12] Kirasić D., Basch D., *Ontology-Based Design Pattern Recognition*, Knowledge-Based Intelligent Information and Engineering Systems, Zagreb, Croatia, 2008.

[13] Kobyliński A., *Ewolucja norm jakości produktu programowego*, Info XXI, Roczniki Kolegium Analiz Ekonomicznych, 36/2015, 91–101, Warszawa, 2015.

[14] Mehlitz P.C., Penix J., *Design for Verification Using Design Patterns to Build Reliable Systems*, 2003.

[15] Nicholson J. i inni, *Automated verification of design patterns: A case study*, Science of Computer Programming, 80 (Part B), 211–222, Elsevier, 2013.

[16] Rasool G., *Customizable Feature based Design Pattern Recognition Integrating Multiple Techniques*, Dysertacja Doktorska, Technische Universitat Ilmenau, Ilmenau, 2010.

[17] Roman A., *Testowanie i jakość oprogramowania. Metody, narzędzia, techniki*, Wydawnictwo Naukowe PWN, Warszawa, 2015.

[18] Singh Rao R., Gupta M., *Design Pattern Detection by Sub Graph Isomorphism Technique*, International Journal of Engineering and Computer Science, 2, 11, 3101–3105, 2013.

[19] Stencel K., Węgrzynowicz P., *Implementation Variants of the Singleton Design Pattern*, 2008.

[20] Tsantalis N. i inni, *Design Pattern Detection Using Similarity Scoring*, IEEE Transactions on Software Engineering, 32, 11, 896–908, 2006.

[21] Vernazza T., Granatella G., Succi G., Benedicenti L., Mintchev M., *Defining metrics for software components*, Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics, Orlando, USA, 2000.

[22] Wojszczyk R., *Pozyskiwanie struktury obiektowej z kodu zarządzanego przy wykorzystaniu metod inżynierii odwrotnej*, Inżynieria oprogramowania: badania i praktyka, Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego, s. 199–213, Warszawa 2014.

[23] Wojszczyk R., *Quality Assessment of Implementation of Strategy Design Pattern*, Advances in Intelligent Systems and Computing, Springer International Publishing, 620, 37–44, 2018.

[24] Wojszczyk R., *The Experiment with Quality Assessment Method Based on Strategy Design Pattern Example*, Advances in Intelligent Systems and Computing, Springer International Publishing, 656, 103–112, 2018.

[25] Wojszczyk R., *Verification of accuracy and cost of use methods of quality assessment of implementation of design patterns*, Applied Computer Science, Polskie Towarzystwo Promocji Wiedzy, Lublin, 15, 1, 2019.

[26] Wojszczyk R., Ratuszniak P., *Trudności w implementacji wzorców projektowych w małych zespołach programistycznych*, Przedsiębiorstwo we współczesnej gospodarce – teoria i praktyka, Wydawnictwo Politechniki Gdańskiej, Gdańsk, 2, 189–201, 2017.