Rafał Wojszczyk
Wydział Elektroniki i Informatyki
Politechnika Koszalińska
rafal.wojszczyk@tu.koszalin.pl

Koncepcja hybrydowej metody do oceny jakości zaimplementowanych wzorców projektowych

Słowa kluczowe: wzorce projektowe, analizowanie oprogramowania, ocena jakości oprogramowania, metryki oprogramowania

1. Wstęp

Poziom życia ludzi poprawia się wraz z rozwojem technologii a w tym elektroniki. Rozwój elektroniki wiąże się z rozwojem oprogramowania, które steruje bądź współpracuje z ową elektroniką. W konsekwencji czego rozwój oprogramowania wpływa również na poziom życia ludzi. Zatem aby ten był jak najwyższy konieczne jest zapewnienie wysokiej jakości oprogramowania, co często jest drugorzędnym czynnikiem decydującym o atrakcyjności danego przedmiotu.

Badania nad wyznaczeniem jakości różnych przedmiotów są prowadzone od wielu lat. W przypadku przedmiotów materialnych sytuacja wydaje się być prostsza, ponieważ podstawowe właściwości fizyczne łatwo zmierzyć. Oprogramowanie jest tworem niematerialnym co przyczynia się do wzrostu trudności pomiarów oprogramowania [13].

Celem artykułu jest przedstawienie autorskiej propozycji metody do oceny jakości zaimplementowanych wzorców projektowych, które w dużym stopniu wpływają na wewnętrzną jakość oprogramowania.

2. Jakość oprogramowania i wzorce projektowe

2.1. Jakość oprogramowania

W wielu źródłach w tym w [6] oraz [14] jako najpopularniejsza definicja jakości oprogramowania przywoływane jest stwierdzenie braku błędów w oprogramowaniu. Takie stwierdzenie dotyczy wyłącznie bardzo wąskiego zakresu cech oprogramowania. Mnogość trudno mierzalnych cech oprogramowania sprawia, iż wspomniana definicja jest stanowczo niewystarczająca aby zastosować ją do współczesnego oprogramowania.

Pierwsze miary oprogramowania dotyczyły łatwomierzalnych cech np. ilość wierszy kodu. Z czasem owe miary zostały połączone z ilością błędów występujących na jeden tysiąc wierszy kodu, czy też ilość wykrytych błędów w określonym czasie działania [7]. Ewolucja paradygmatów programowania, zastosowania oprogramowania, dodatkowo badania nad jakością oprogramowania doprowadziły do powstania całościowych modeli jakości oprogramowania: CUMPRIMDA stworzone przez IBM lub FRUPS stworzone przez Hewlett-Packard [7]. Obie metody pozwalają na wyrażenie jakości oprogramowania w kilku kategoriach. Kategorie często są niemierzalne bezpośrednio, dlatego na wyrażenie każdej z nich składa się kilka charakterystyk, które są wyznaczane na podstawie wartości odpowiednich miar. Jak zauważono w [10] taki podział na kategorie bądź charakterystyki, a następnie na podcharakterystyki oraz ostatecznie na miary, jest typowy dla wielu metod dotyczących jakości oprogramowania. Kategorie w przypadku FURPS to [13]: funkcjonalność, łatwość użytkowania, niezawodność, efektywność, wsparcie. Natomiast model CUPRIMDA zawiera kategorie [7]: możliwości, łatwość obsługi, wydajność, niezawodność, łatwość instalacji, łatwość zarządzania i serwisu, dokumentacja, dostępność. Można zauważyć, iż większość z wymienionych kategorii nie dotyczy miar jakości kodu źródłowego.

Kolejnym przykładem kompleksowej metody opisującej jakość oprogramowania jest norma ISO/IEC 9126 z 2001 roku z dodatkowymi częściami wydanymi w latach 2002-2004. W odróżnieniu od dwóch wcześniej wspomnianych modeli jakości, norma ISO/IEC 9126 zakłada bardziej ogólny podział podstawowych części modelu: część 1 to ogólny model jakości, część 2 zawiera miary zewnętrzne, część 3 zawiera miary wewnętrzne, część 4opisuje miary jakości użytkowej. Na większą uwagę zasługuje charakterystyka konserwowalności (od ang. mainanability) wynikająca z trzeciej części normy. Na charakterystykę konserwowalności składają się następujące podcharakterystyki:

- podatność na analizę (od ang. analysability) to miary, które pozwalają przewidzieć potrzebny wysiłek oraz zasoby do zdiagnozowania niedoskonałości oprogramowania, zidentyfikowania przyczyn błędów w oprogramowaniu lub jego zmodyfikowanych częściach,
- metryka zmienności (od ang. changeability) pozwala przewidzieć potrzebny trud do wprowadzenia modyfikacji w oprogramowaniu,
- stabilność (od ang. stability) to miary zdolności oprogramowania do uniknięcia niespodziewanych modyfikacji,
- metryki testowalności (od ang. testability) odpowiadają za ilość zaprojektowanych oraz zaimplementowanych funkcji do testowania oprogramowania,

- zgodność ze standardami konserwacji (od ang. maintainability compliance) to metryka opisująca wymóg przestrzegania standardów, konwencji oraz regulacji prawnych odnoszących się do utrzymania i konserwacji oprogramowania.

Pewną nieścisłością jest brak polskiego tłumaczenia tej normy, z tej przyczyny w literaturze można spotkać różne propozycje tłumaczeń, również inne od zaproponowanych powyżej.

Bazując na wymienionej definicji zmienności z modelu ISO/IEC 9126 można przedstawić rozwiązania, które będą sprzyjały pozytywnym wartościom tej podcharakterystyki. Jednym z najpopularniejszych rozwiązań tego problemu są wzorce projektowe opisane przez [4]. W [4] opisanych zostało 8 różnych problemów programistycznych, w których wykorzystanie wzorców projektowych sprzyja uniknięciu problemów związanych ze zmianami oprogramowania. Ponadto zastosowanie wzorców projektowych może też wpływać pozytywnie na podcharakterystykę stabilności.

2.2. Wzorce projektowe

Wzorce projektowe opisane w [4] to szablony gotowych mechanizmów, które można wykorzystać do rozwiązania typowych problemów pojawiających się cyklicznie w projektowaniu i programowaniu obiektowym. Jednakże nie są to gotowe rozwiązania, ponieważ wykorzystanie każdego wzorca wymaga jego odpowiedniej implementacji zgodnie z kontekstem oprogramowania.

Autorzy [4] dokonali klasyfikacji oraz szczegółowego opisu wzorców projektowych. Autorami samych wzorców projektowych jest ogół społeczeństwa, które wówczas skupiało się nad oprogramowaniem obiektowym. Aktualnie pomimo upływu 20 lat wzorce projektowe nadal uważane są za przejaw dobrych praktyk, język komunikacji między twórcami oprogramowania oraz symbol wysokiej jakości kodów źródłowych oprogramowania.

Wykorzystanie w oprogramowaniu wzorców projektowych niesie za sobą bardzo wiele korzyści, więc wysoko pożądanym wydaje się być zautomatyzowanie procesów ich implementacji oraz weryfikacji. Niestety, jest to znacząco utrudnione z kilku powiązanych ze sobą powodów:

- jest to twór społeczeństwa, który wynika z praktyki oraz doświadczenia twórców oprogramowania, z tej przyczyny wzorce projektowe nie podlegały żadnym regulacją prawnym czy restrykcjom narzuconym odgórnie,
- brak standaryzacji nawet dla dobrze znanych oraz sprawdzonych wzorców,
- brak formalnej kontroli oraz idea wykorzystania wzorców za każdym razem nieco inaczej [3] sprawia, że każdy deweloper może zaimplementować

dany wzorzec w pewnym zakresie dowolności, czego efektem jest znacząca różnorodność implementacji wzorców projektowych,

- różnorodność języków programowania, które same w sobie mogą realizować niektóre cechy wzorców projektowych dodatkowo potęguje różnorodność implementacji.

3. Badania dotyczące wzorców projektowych

Wzorce projektowe zyskały bardzo dużą popularność wśród programistów tj. praktyków w rzemiośle programowania. Zatem nie powinno nikogo dziwić, iż jest to również przedmiot badań dla wielu naukowców. W przypadku pierwszej grupy osób wystarczającym sposobem reprezentacji wzorców projektowych jest opis nieformalny, który został wykorzystany w [4]. Reprezentacja nieformalna jest dobrym sposobem do nauki oraz przedstawienia ogólnych założeń i cech wzorców. W naukowym podejściu konieczne jest wprowadzenie pewnego formalizmu, aby interpretacja wyników badań nie budziła żadnych wątpliwości znaczeniowych.

W badaniach nad wzorcami projektowymi pojawia się wiele różnych problemów, od problemów związanych z ewolucją i tworzeniem wzorców, po procesy wyszukiwania, weryfikacji czy też samoistnej implementacji. Dalsza uwaga została skupiona na procesie weryfikacji wzorców projektowych. Jednakże problem wyszukiwania wystąpień wzorców projektowych w kodzie źródłowym jest bardzo istotny, ponieważ spełnienie określonych kryteriów wyszukiwania przez badany fragment oprogramowania w wielu przypadkach pokrywa się z przeprowadzeniem podstawowej weryfikacji. Prace związane z tym zagadnieniem to [15], gdzie za podstawowy model wyszukiwania wzorców zostało wykorzystane izomorficzne wyszukiwanie podgrafów w grafie, w oparciu o grafy zostało to zrealizowane również w [16], natomiast w [9] została wykorzystana ontologia.

Samo zagadnienie weryfikacji poprawności zaimplementowanych wzorców projektowych wydaje się być mniej popularne niż wyszukiwanie ich wystąpień, świadczy o tym mniejsza ilość prac poświęconych bezpośrednio tej tematyce. Pośród powiązanych prac można wyróżnić dominującą grupę, w których podstawowe modele wywodzą się z opisu regułowego. Efektem tychże prac są badania nad stworzeniem specjalnych języków: SOUL[2] bądź SPINE[1], jednakże już sami autorzy wskazują pewne ograniczenia proponowanych przez siebie metod. W [11] wykorzystany został rachunek lambda wraz z wprowadzeniem dekompozycji wzorców projektowych na dodatkowe warstwy. Za drugą grupę prac można uznać prace, w których wykorzystywane są istniejące metryki oprogramowania lub proponowane są ich rozwinięcia [8]. Inny poruszany wątek to zależności pomiędzy wynikami metryk a ilością zaimplementowanych wzorców w badanym oprogramowaniu [12] oraz [5].

4. Opis metody

4.1. Ogólny zarys metody

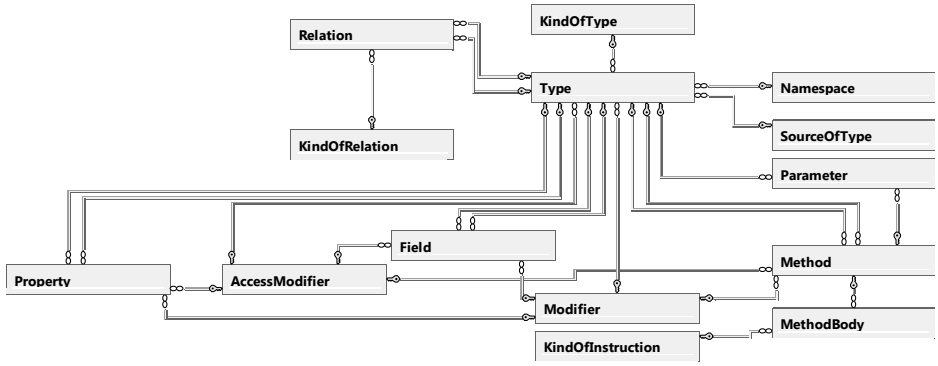
Statyczną analizę oprogramowania można przedstawić w trzech ogólnych modułach:

1. Akwizycja danych to pozyskanie informacji o artefaktach w oprogramowaniu a następnie zapisanie ich w postaci formalnej. Źródłem danych wejściowych może być bezpośrednio kod źródłowy co wymaga jego interpretacji lub kod pośredni co zostało przedstawione w [17].
2. Analiza danych to odpowiednie przebadanie danych pozyskanych z poprzedniego modułu, gdzie elementem łączącym oba moduły jest model danych w postaci formalnej. Rodzaj wykonywanych analiz zależy od postawionego wcześniej celu. Wyniki analizy powinny zostać opisane ponownie w formalnym modelu danych.
3. Interpretacja wyników to przetworzenie uzyskanych wyników analizy z drugiego modułu metody w celu łatwiejszego zrozumienia ich przez człowieka.

Trójmodułowa budowa całej metody pozwala na osiągnięcie dość dużej uniwersalności, jest to możliwe dzięki zastępowaniu wybranych modułów innymi modułami (np. inny rodzaj akwizycji danych). Istotne jest przy tym zachowanie formalnych i dobrze udokumentowanych elementów łączących moduły. W przypadku obu połączeń jest to model danych, który został zaproponowany w [17]. Model zawiera w sobie podstawowy model danych reprezentujący strukturę obiektową oprogramowania, co wywodzi się z podstawowych założeń paradygmatu programowania obiektowego. Model ten został nazwany modelem produktu, ponieważ zawiera w sobie pozyskane artefakty z oprogramowania tj. badanego produktu.

4.2. Model produktu

Model produktu, a dokładniej opis struktury obiektowej, którą zawiera badane oprogramowanie, jest uzupełniany podczas akwizycji danych. Model ten został przedstawiony na rysunku nr 1. Jest to rozwinięcie koncepcji formalnego opisu oprogramowania przedstawionego w [17]. Elementy tego rozwinięcia to zwiększenie szczegółowości danych w przypadku: zawartych instrukcji w ciele metod (MethodBody), zidentyfikowanie przestrzeni nazw (Namespace) do jakiej należy przechowywany typ oraz wskazanie źródła pochodzenia typu (SourceOfType) np.: typ własny, systemowy, zewnętrzny, zaślepkowy.



Rys. 1. Diagram modelu produktu

Mnogość relacji na diagramie z rysunku 1, a szczególnie powiązań z tabelą Type reprezentującą dostępne typy, może budzić wątpliwości oraz utrudniać czytanie diagramu. Relacje, w których tabela Type przechowuje klucze obce innych tabel są wymagane w celu zagregowania występujących typów w odpowiednich kategoriach. Pozostałe relacje wynikają bezpośrednio z opisu wywodzącego się z paradygmatu programowania obiektowego, tj. typ może zawierać pola, metody itd. W kilku przypadkach występują również podwójne relacje między tabelami. Wynika to z faktu, że elementy składowe też reprezentują konkretny typ, np. pole w klasie jest instancją innej klasy z badanego oprogramowania. Ponowne wykorzystanie typów również jest cechą paradygmatu programowania obiektowego. Modyfikatory dostępu oraz pozostałe modyfikatory określające pewne cechy rozważanego elementu, np. oznaczenie jako statyczny, muszą być skojarzone z elementami składowymi typów oraz z samym typem, a to dodatkowo zmniejsza czytelność diagramu.

4.3. Etapy analizy

Użycie słowa "hybrydowa" w zaproponowanej nazwie metody wywodzi się od trzech różnych sposobów weryfikacji wzorców projektowych. Sposoby te zostały zrealizowane w trzech etapach analizy a każdy z etapów zawiera podcharakterystyki, które przyczyniają się do wyniku końcowego. W dalszej części podrozdziału znajduje się koncepcyjny opis tych etapów.

Dane wejściowe dla metody to opisany wcześniej model produktu. Dodatkowo wymagane jest aby w tym modelu był wskazany rdzeń lub główny element badanego artefaktu (dla każdego przypadku wystąpienia). Przykładem tego może być klasa zawierają implementację wzorca Singleton czy też interfejs zawierający deklarację wymaganej funkcji we wzorcu Strategia. To ograniczenie powoduje, że poddane analizie będą wyłącznie dane powiązane z badanym artefaktem.

Pierwszym etapem analizy jest weryfikacja względem opisu regułowego, polega to na sprawdzeniu czy zostały spełnione podstawowe reguły w badanym artefakcie. Celem tego jest wykazanie, że badany fragment oprogramowania zawiera w sobie najważniejsze cechy implementacji oczekiwanego wzorca projektowego. W prostym przypadku wzorca Singleton tymi regułami są: prywatny konstruktor, publiczna i statyczna właściwość udostępniająca instancję Singletonu. Naturalnym jest, że dla każdego wzorca projektowego należy przewidzieć osobne zbiory wymaganych reguł.

W drugim etapie następuje porównanie zgodności z modelem referencyjnym, tj. zestawienie struktury obiektowej badanego artefaktu z abstrakcyjną strukturą obiektową opisaną w modelu referencyjnym. Dla każdego wzorca projektowego może istnieć wiele różnych wariantów jego implementacji, które należy opisać w modelu referencyjnym a porównanie należy wykonać z najlepiej dopasowanym wariantem. Celem tego porównania jest wykazanie bardzo szczegółowych cech wzorców projektowych. W prostym przykładzie wzorca Singleton porównaniu może podlegać ilość oraz rodzaj występujących relacji pomiędzy klasą zawierającą wzorzec a innymi elementami badanego oprogramowania.

Wyznaczenie wartości metryk to uzupełnienie wyników cząstkowych uzyskanych we wcześniejszych etapach metody o dodatkowe charakterystyki. Wyniki metryk są interpretowane w odpowiedni sposób dla każdego wzorca projektowego. Jest to kontynuacja idei przedstawionej w [18]. Podstawowe przewidziane metryki to zestawy CK, MOOD i Roberta Martina, jednakże mogą zostać wykorzystane również inne metryki w zależności od badanego artefaktu. Przykładem dla wzorca Singleton jest zalecany wynik 0 dla metryk DIT oraz NOC, gdzie w ogólnym przypadku zalecana wartość tej metryki to nie więcej niż 6 [18].

Na każdym z powyższych etapów analizy występuje wartościowanie uzyskanych wyników podcharakterystyk, tj. każdemu uzyskanemu wynikowi przypisywany jest odpowiedni współczynnik wagi. Oprócz tego przypisane są dodatkowe informacje: opcjonalność wystąpienia badanej cechy, wzajemne wykluczanie się, tożsamość jednej cechy z drugą, hierarchie ważności. Wagi początkowo są ustalone na podstawie rozważań teoretycznych, następnie mogą być poddane ewaluacji w celu ich polepszenia. Wykazanie zachodzących korelacji pomiędzy wagami, wynikami czy też samymi charakterystykami tworzy dodatkową przestrzeń do rozwoju zaproponowanej metody.

5. Dalszy rozwój metody

Przewidywane dalsze prace dotyczące rozwoju modułu analizy przewidują opracowanie szczegółowego modelu referencyjnego, który będzie brał udział w każdym etapie analizy. W celu skutecznego połączenia wszystkich etapów analizy konieczne jest opracowanie zachodzących zależności pomiędzy badanymi

cechami: opcjonalność wystąpienia badanej cechy, hierarchie ważności, tożsamość jednej cechy z drugą czy też wzajemne wykluczanie się.

Postępy w realizacji modułu analizy są konieczne aby przystąpić do prac badawczych nad interpretacją wyników, dlatego na tym etapie nie można jeszcze określić szczegółowych dalszych prac. Mimo to dość oczywistym kierunkiem wydają się prace badawcze dotyczące interpretacji wyników charakterystyk czy też badanie korelacji pomiędzy wynikami i wagami, w celu zaproponowania ostatecznej oceny jakości badanych wzorców projektowych zgodnej z opinią eksperta.

Moduł akwizycji danych również wymaga dalszego rozwoju. Model produktu powinien umożliwiać między innymi wskazanie rdzenia badanego wzorca projektowego.

6. Podsumowanie

W pracy zaprezentowano koncepcję metody umożliwiającej ocenę jakości zaimplementowanych wzorców projektowych. Przedstawiono również ogólne podejście do oceny oprogramowania w tym charakterystykę konserwowalności z normy ISO 9126:2000.

Proponowana metoda składa się z trzech modułów: akwizycji danych, analizy, interpretacji wyników. W pierwszym module badane oprogramowanie zostaje przetworzone na model produktu. Analiza jest wykonywana w trzech etapach bazujących na opisie regułowym, modelu referencyjnym oraz wyliczaniu metryk. Na podstawie uzyskanych wyników analizy następuje ocena jakości badanego wzorca projektowego. Dalszy rozwój metody związany jest głównie pracami nad dwoma ostatnimi modułami. Ponadto mając na uwadze w dalszych pracach zachowanie dużej uniwersalności metody możliwe będzie analizowanie nie tylko wzorców projektowych ale też innych artefaktów programowania obiektowego.

Bibliografia

1. Blewitt A., *HEDGEHOG: Automatic Verification of Design Patterns in Java*, Dysertacja doktorska, University of Edinburgh, 2006.
2. Fabry J., Mens T., *Language-Independent Detection of Object-Oriented Design Patterns*, w: Journal Computer Languages: Systems and Structures, Vol. 30, Issue 1-2, 2004, s. 21-33.
3. Fowler M. i inni, *Architektura systemów zarządzania przedsiębiorstwem – Wzorce projektowe*, Helion, Gliwice, 2005.
4. Gamma E. i inni, *WZORCE PROJEKTOWE Elementy oprogramowania wielokrotnego użytku*, Helion, Gliwice, 2010.

5. Hernandez J. i inni, *Selection of Metrics for Predicting the Appropriate Application of design patterns*, 2nd Asian Conference on Pattern Languages of Programs, 2011.
6. Jureczko M., *Metody zarządzania zapewnianiem jakości oprogramowania wykorzystujące modele predykcji defektów*, Dysertacja doktorska, Politechnika Wrocławska, Wrocław, 2011.
7. Kan S. H., *Metryki i modele w inżynierii jakości oprogramowania*, PWN SA, Warszawa, 2006.
8. Khaer Md. A. i inni, *An Empirical Analysis of Software Systems for Measurement of Design Quality Level Based on Design Patterns*, w: Computer and information technology, IEEE, 2007.
9. Kirasić D., Basch D., *Ontology-Based Design Pattern Recognition*, w: Knowledge-Based Intelligent Information and Engineering Systems, Zagreb, Croatia, 2008.
10. Kobyliński A., *ISO/IEC 9126 – analiza modelu jakości produktów programowych*, w: Systemy Wspomagania Organizacji 2003, red. Porębska-Miąc T., Sroka H., Prace naukowe Akademia Ekonomiczna w Katowicach, 2003.
11. Krishnaswami N. R., *Design Patterns in Separation Logic*, w: TLDI'09, ACM, New York, 2009, s. 105-116.
12. Masuda G., Sakamoto N., Ushijima K., *Evaluation and Analysis of Applying Design Patterns*, w: Proceedings of the International Workshop on., 1999.
13. Pressman R.S., *Inżynieria oprogramowania, praktyczne podejście do inżynierii oprogramowania*, WNT, Warszawa 2004.
14. Protasowicki T., Stanik J., *Symulacyjne badanie jakości oprogramowania w zwinnym procesie produkcji*, Inżynieria oprogramowania: badania i praktyka, Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego, 2014, s. 157-178.
15. Singh Rao R., Gupta M., *Design Pattern Detection by Greedy Algorithm Using Inexact Graph Matching*, International Journal Of Engineering And Computer Science, Volume 2 Issue 10, 2013, s. 3658-3664.
16. Tsantalis N. i inni, *Design Pattern Detection Using Similarity Scoring*, w: IEEE Transactions on Software Engineering, Volume: 32, Issue: 11, 2006, s. 896-908.
17. Wojszczyk R., *Pozyskiwanie struktury obiektowej z kodu zarządzanego przy wykorzystaniu metod inżynierii odwrotnej*, w: Inżynieria oprogramowania: badania i praktyka, Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego, 2014, s. 201-216.

18. Wojszczyk R., *Zestawienie metryk oprogramowania obiektowego opartych na statycznej analizie kodu źródłowego*, w: Zarządzanie projektami i modelowanie procesów, Zeszyty Rady Naukowej Polskiego Towarzystwa Informatycznego, 2013, s. 95-107.

Streszczenie

Wzorce projektowe są jednym z elementów, które wpływają pozytywnie na wewnętrzną jakość oprogramowania. Celem publikacji jest przedstawienie koncepcji metody umożliwiającej ocenę zaimplementowanych wzorców projektowych. W pracy krótko przedstawiono różne modele jakości oprogramowania oraz wybrane badania związane z wzorcami projektowymi. Następnie opisana została koncepcja proponowanej metody, która łączy w sobie różne podejścia do statycznej analizy oprogramowania. Przedstawione zostało również rozwinięcie modelu danych reprezentującego badane oprogramowanie oraz kierunki dalszego rozwoju metody.

Abstract

Design patterns are one of the elements that have a positive impact on the internal quality of software. The aim of the publication is to present the concept of a method for evaluation of the implemented design patterns. This paper briefly presents the various models of software quality and selected research related to design patterns. Then it describes the concept of the proposed method which combines different approaches to static software analysis. It has also been presented a data model describing examined the software and directions for the further development of the method.

Keywords: design patterns, software analysis, evaluation of software quality, software metric