Czysty kod w PHP

Wskazówki ekspertów i najlepsze rozwiązania pozwalające pisać piękny, przystępny i łatwy w utrzymaniu kod PHP

ALEXANDRE DAUBOIS | CARSTEN WINDLER

Tytuł oryginału: Clean Code in PHP: Expert tips and best practices to write beautiful,
human-friendly, and maintainable PHP

Tłumaczenie: Krzysztof Bąbol

ISBN: 978-83-8322-718-4

Copyright © Packt Publishing 2022. First published in the English language
under the title 'Clean Code in PHP – (9781804613870)'

Polish edition copyright © 2023 by Helion S.A.

All rights reserved. No part of this book may be reproduced or transmitted in any
form or by any means, electronic or mechanical, including photocopying, recording
or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu
niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii
metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym,
magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi
bądź towarowymi ich właścicieli.

Autor oraz wydawca dołożyli wszelkich starań, by zawarte w tej książce informacje
były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich
wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych
lub autorskich. Autor oraz wydawca nie ponoszą również żadnej odpowiedzialności
za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/phpkod
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści

O autorze 11

O recenzentach 12

Wstęp 13

CZĘŚĆ 1. Prezentacja czystego kodu

ROZDZIAŁ 1
Co to takiego czysty kod i dlaczego powinno Ci na nim zależeć? 19
- O czym będzie ta książka? 20
- Wyjaśnienie, czym jest czysty kod 21
- Znaczenie czystego kodu dla zespołów 22
- Znaczenie czystego kodu we własnych projektach 24
- Podsumowanie 25

ROZDZIAŁ 2
Kto może decydować o tym, jakie są „dobre praktyki”? 26
- Kto w ogóle decyduje o tych sprawach? 27
- Najlepsze rozwiązania — skąd tak naprawdę się biorą? 28
  - Zasady wzorców projektowych 28
- Świadomość sytuacji 33
- Zachowaj konsekwencję, a szybciej uzyskasz rezultaty 35
  - O narzędziach do analizy kodu 36
  - O testowaniu i jego wielu formach 37
- Podsumowanie 39

ROZDZIAŁ 3
Programuj, nie rób akrobacji 41
- Czym jest kod? 41
  - Trochę historii 42
  - Przeznaczenie kodu 42

Pisz zrozumiale, a nie sprytnie 43
Uwagi na temat możliwości utrzymania kodu 45
Używanie operatorów binarnych i zapisu ósemkowego, szesnastkowego i dwójkowego 46
Nadawanie wartości zmiennym i stosowanie instrukcji goto 46
Przesadne komentowanie 47
Korzystanie z operatorów trójargumentowych 47
Stosowanie skrótów 48
Wprowadzanie w kodzie mikrooptymalizacji 48
Programowanie na nowo metod z biblioteki SPL 48
Podsumowanie 49

ROZDZIAŁ 4
Tu chodzi o coś więcej niż sam kod 51
PHP jako ekosystem 51
Wybór właściwych bibliotek 53
Parę słów o wersjonowaniu semantycznym 55
Czym jest wersjonowanie semantyczne? 56
Jak sobie radzić z wersjonowaniem semantycznym 57
Stabilność kontra trendy 58
Podsumowanie 60

ROZDZIAŁ 5
Optymalizacja czasu pracy i rozdzielenie odpowiedzialności 61
Konwencje nazewnicze i organizacyjne 61
Pliki klas i interfejsów 62
Pliki wykonywalne 62
Elementy zawartości i zasoby sieci WWW 62
Nazewnictwo klas, interfejsów i metod 63
Nazewnictwo folderów 63
Rozdzielenie odpowiedzialności 64
Rozsyłanie zdarzeń 65
Objaśnienie polimorfizmu — interfejsy i klasy abstrakcyjne 67
Interfejsy 67
Klasy abstrakcyjne 68
Podsumowanie 69

ROZDZIAŁ 6

PHP ewoluuje — deprecjacje i rewolucje 71
Nowe wersje PHP w porównaniu ze starymi 71
Ścisła kontrola typów 72
Raportowanie błędów 72
Atrybuty 73
Przełom w wersji 8 74
Konstrukcja match 74
Argumenty nazwane 75
Klasy i właściwości tylko do odczytu 75
Migrowanie zasobów do odpowiednich klas 76
Ochrona wrażliwych argumentów przed wyciekiem 78
Podsumowanie 79

CZĘŚĆ 2. Utrzymywanie jakości kodu

ROZDZIAŁ 7

Narzędzia jakości kodu 83
Wymagania techniczne 83
Sprawdzanie składni i stylu kodu 84
Linter wbudowany w PHP 84
PHP CS Fixer: szperacz kodu 85
Statyczna analiza kodu 90
phpcpd — wykrywacz kopiowania i wklejania kodu 90
PHPMD: wykrywacz bałaganu w PHP 92
PHPStan — analizator statyczny dla języka PHP 98
Psalm — maszyna lintująca do analizy statycznej dla PHP 104
Rozszerzenie środowisk IDE 108
PHP Inspections (EA Extended) 110
Intelephense 113
Podsumowanie 115
Materiały dodatkowe 116

ROZDZIAŁ 8

Wskaźniki jakości kodu 117
Wymagania techniczne 117
Prezentacja wskaźników jakości kodu 118
Aspekty jakości oprogramowania 118
Wskaźniki jakości kodu 119

Kup ksi k Pole ksi k

Zbieranie wskaźników w PHP 123
phploc 124
PHP Depend 126
PhpMetrics 128
Zalety i wady korzystania ze wskaźników 135
Zalety 135
Wady 136
Podsumowanie 137
Materiały dodatkowe 137

ROZDZIAŁ 9
Organizacja narzędzi jakości PHP 138
Wymagania techniczne 138
Instalowanie narzędzi jakości kodu przy użyciu menedżera Composer 138
Instalowanie narzędzi jakości kodu z wykorzystaniem sekcji require-dev 139
Instalacja globalna 141
Skrypty menedżera Composer 142
Instalowanie narzędzi jakości kodu jako plików phar 143
Utrzymywanie plików phar w porządku 144
Zarządzanie plikami phar przy użyciu programu Phive 145
Dodawanie menedżera Phive do projektu 147
Podsumowanie 147

ROZDZIAŁ 10
Testowanie automatyczne 149
Wymagania techniczne 150
Dlaczego potrzebujesz testów automatycznych? 150
Łatwiejsza refaktoryzacja dzięki testom 152
Typy testów automatycznych 153
Testy jednostkowe 153
Testy integracyjne 157
Testy E2E 159
Piramida testowa w praktyce 161
Uwagi o pokryciu kodu 162
Zapoznanie z pokryciem kodu 162
Jak generować raporty o pokryciu kodu 163
Korzystanie z adnotacji @covers 167
Co testować? 167
Podsumowanie 169
Materiały dodatkowe 169

Kup ksi k Pole ksi k

ROZDZIAŁ 11

Ciągła integracja ... 171
Wymagania techniczne ... 171
Dlaczego ciągła integracja jest potrzebna? ... 172
Koszty błędu ... 172
Jak zapobiegać błędom ... 173
Prezentacja ciągłej integracji ... 174
Potok budowy ... 175
Etap 1. — budowanie projektu ... 176
Etap 2. — analiza kodu ... 178
Etap 3. — testy ... 179
Etap 4. — wdrożenie ... 180
Integrowanie potoku z przepływem pracy ... 182
Budowanie potoku w narzędziu GitHub Actions ... 182
GitHub Actions w skrócie ... 183
Etap 1. — budowanie projektu ... 184
Etap 2. — analiza kodu ... 187
Etap 3. — testy ... 189
Etap 4. — wdrożenie ... 190
Integrowanie potoku z własnym przepływem pracy ... 191
Twój lokalny potok — haki narzędzia Git ... 194
Konfigurowanie haków narzędzia Git ... 195
Haki narzędzia Git w praktyce ... 197
Zaawansowane użycie ... 199
Dygresja — wprowadzanie ciągłej integracji do istniejącego oprogramowania ... 200
Krok po kroku ... 201
Spojrzenie na ciągłe dostarczanie ... 202
Podsumowanie ... 203
Materiały dodatkowe ... 204

ROZDZIAŁ 12

Praca w zespole ... 205
Wymagania techniczne ... 205
Standardy pisania kodu ... 206
Podążanie za istniejącymi standardami ... 206
Zasady pisania kodu ... 209
Przykładowe zasady pisania kodu ... 210
Ustalanie zasad ... 217
Przeglądy kodu ... 218
Dlaczego należy dokonywać przeglądów kodu ... 219
Co powinny obejmować przeglądy kodu? ... 219

Najlepsze rozwiązania dotyczące przeglądów kodu ...220
Zapewnienie przeprowadzania przeglądów kodu ...222
Definicja ukończenia ...223
Wnioski na temat przeglądów kodu ...224
Wzorce projektowe ...224
Zapoznanie ze wzorcami projektowymi ...224
Wzorce projektowe często spotykane w języku PHP ...225
Antywzorce ...236
Podsumowanie ...238
Materiały dodatkowe ...239

ROZDZIAŁ 13
Tworzenie efektywnej dokumentacji ...240
Wymagania techniczne ...240
Dlaczego dokumentacja ma znaczenie? ...240
Dlaczego dokumentacja jest ważna? ...241
Dokumentacja dla programistów ...242
Tworzenie dokumentacji ...243
Dokumenty tekstowe ...243
Diagramy ...244
Generatory dokumentacji ...246
Dokumentacja w kodzie źródłowym ...251
Adnotacje nie są kodem ...251
Nieczytelny kod ...251
Nieaktualne komentarze ...252
Bezużyteczne komentarze ...252
Błędne lub nieprzydatne sekcje DocBlock ...253
Komentarze TODO ...253
Kiedy komentowanie jest przydatne? ...253
Testy jako dokumentacja ...254
Podsumowanie ...255
Materiały dodatkowe ...255

Kup ksi k Pole ksi k

Wskaźniki jakości kodu

Rozdział 8

Czy nie byłoby wspaniale, gdybyśmy mogli mierzyć jakość naszego oprogramowania? Programiści często chcą bez końca poprawiać swoje programy — ale może są one już „wystarczająco dobre”? Skąd możemy wiedzieć, czy oprogramowanie osiągnęło odpowiedni stan?

Wskaźniki jakości oprogramowania zostały wprowadzone przez pewnych mądrych ludzi we wczesnym okresie rozwoju informatyki. Rozważali ten temat w latach 70. ubiegłego wieku i opracowali pomysły, które są w użyciu po dziś dzień. My oczywiście będziemy chcieli skorzystać z tej wiedzy i zastosować ją w naszych własnych projektach.

W tym rozdziale poruszymy następujące zagadnienia:

- Prezentacja wskaźników jakości kodu.
- Zbieranie wskaźników w PHP.
- Zalety i wady korzystania ze wskaźników.

Wymagania techniczne

Jeśli udało Ci się przebrnąć przez poprzedni rozdział i wypróbować znajdujące się tam narzędzia, masz już prawdopodobnie zainstalowane wszystko, co potrzebne do prześledzenia tego rozdziału. Jeśli nie, zrób to koniecznie przed uruchomieniem pokazanych tutaj przykładów.

Kod źródłowy przykładów z tego rozdziału znajduje się w archiwum dostępnym pod adresem *https://ftp.helion.pl/przyklady/phpkod.zip*, w folderze *r08*.

Kup ksi k Pole ksi k

Prezentacja wskaźników jakości kodu

W tym podrozdziale dowiesz się, jak w ogóle mierzy się jakość oprogramowania. Przyjrzymy się niektórym najczęściej używanym w świecie PHP wskaźnikom i wyjaśnimy, co mówią na temat kodu, jak się je zbiera oraz czy są przydatne, czy też nie.

Aspekty jakości oprogramowania

Zanim przejdziemy do liczb, musimy najpierw wyjaśnić coś ważnego: co tak naprawdę oznacza jakość oprogramowania? Z pewnością każdy do pewnego stopnia rozumie, czym jest jakość, ale może mieć trudność z wyrażeniem tego własnymi słowami. Na szczęście istnieją modele, takie jak **FURPS**, opracowany w firmie Hewlett-Packard jeszcze w latach 80. ubiegłego wieku. Ten akronim pochodzi od słów:

- ***Functionality*** (funkcjonalność): Czy oprogramowanie ma szeroki zakres zastosowań? Czy zostało opracowane z myślą o bezpieczeństwie?
- ***Usability*** (użyteczność): Czy oprogramowanie jest wygodne w obsłudze? Czy ma dokumentację i łatwo zrozumieć, o co w nim chodzi?
- ***Reliability*** (niezawodność): Czy oprogramowanie jest cały czas dostępne? Na ile prawdopodobne są awarie albo błędy mające wpływ na wynik działania?
- ***Performance*** (wydajność) wskazuje szybkość oprogramowania: Czy efektywnie wykorzystuje ono dostępne zasoby? Czy dobrze się skaluje?
- ***Supportability*** (wspieralność): Czy oprogramowanie można z powodzeniem testować i pielęgnować? Czy jest łatwe w instalacji i czy można je przetłumaczyć na inne języki (zlokalizować)?

Kolejne aspekty jakościowe to między innymi **dostępność** (ang. *accessibility*) i **zgodność z przepisami** (ang. *legal conformity*). Jak widać, ten model obejmuje takie sprawy jak komfort użytkowania i dokumentacja, którymi my, programiści PHP, zwykle się nie zajmujemy. Dlatego na jakość oprogramowania możemy patrzeć z dwóch różnych punktów widzenia, mówiąc o jakości zewnętrznej i wewnętrznej. Przyjrzyjmy się bliżej temu, co oznaczają te pojęcia:

- **Jakość zewnętrzna** — jej elementami są aspekty zewnętrzne, czyli widziane przez użytkownika. Obejmuje wiele właściwości wymienionych wcześniej. Ich cechą wspólną jest to, że mogą być mierzone bez styczności z kodem i bez jego analizowania — pomyśl o narzędziach sprawdzających wydajność, które mierzą czas odpowiedzi na żądanie, albo o testach **od początku do końca** (ang. *end-to-end*), które emulują działania użytkownika i sprawdzają aplikację w sposób automatyczny.

Kup książkę Poleć książkę

- **Jakość wewnętrzna** — my, programiści, zwykle dbamy bardziej o wewnętrzną jakość oprogramowania. Czy kod jest czytelny i zrozumiały? Czy łatwo go rozbudować? Czy możemy dla niego napisać testy? Co prawda użytkownicy nigdy nie zobaczą kodu, a łatwość jego testowania ich nie obchodzi; ale jakość wewnętrzna ma na nich wpływ pośredni: kod o wysokiej jakości zawiera mniej błędów i często (choć nie zawsze) jest szybszy i bardziej wydajny. Jak wiadomo, jest też łatwiejszy do rozbudowy i utrzymania. Zazwyczaj takie aspekty można sprawdzić za pomocą automatycznych testów jednostkowych albo analizatorów kodu.

W tej książce skupimy się na wewnętrznej jakości kodu. Dlatego właśnie mówimy tylko o jakości kodu i nie posługujemy się szerszym pojęciem jakości oprogramowania.

Wskaźniki jakości kodu

Teraz, gdy lepiej rozumiemy, co oznacza jakość kodu, przyjrzyjmy się temu, jakie są jej wskaźniki. W tym punkcie omówimy następujące zagadnienia:

- liczba wierszy kodu;
- złożoność cyklomatyczna;
- złożoność NPath;
- wskaźniki Halsteada;
- indeks ryzyka zmian w kodzie;
- indeks utrzymywalności kodu.

Liczba wierszy kodu

Liczba **wierszy kodu** (ang. *lines of code*, LOC) w projekcie nie jest wskaźnikiem jakości. Jest jednak przydatnym instrumentem, pozwalającym zorientować się w wielkości projektu — na przykład wtedy, gdy ktoś zaczyna nad nim pracować. Ponadto, jak zobaczymy, jest podstawą do liczenia innych wskaźników. Pozwala też dowiedzieć się, z iloma wierszami kodu mamy do czynienia — chociażby wtedy, gdy trzeba oszacować nakład pracy potrzebny do refaktoryzacji pewnych klas.

Dlatego właśnie liczbie tej warto przyjrzeć się bliżej. Zacznijmy od tego, że można wyróżnić jej następujące warianty:

- **LOC** — wskaźnik ten pozwala po prostu policzyć wszystkie wiersze kodu, łącznie z tymi, które są puste lub zawierają komentarze.
- **Liczba wierszy kodu będących komentarzami** (ang. *Comment Lines of Code*, CLOC) — miara ta mówi, ile z wierszy kodu to komentarze. Może być wskaźnikiem tego, jak dobrze wyjaśniany jest kod. Jednak wiemy, że komentarze

Kup ksi k Pole ksi k

zwykle tracą wartość (szybko stają się nieaktualne i często bardziej szkodzą, niż pomagają), więc nie możemy doradzić, ile powinno ich być, ani podać żadnej innej złotej zasady. Mimo wszystko warto znać ten wskaźnik.

- **Liczba wierszy kodu niebędących komentarzami** (ang. *Non-Comment Lines of Code*, NCLOC) — jeśli chcesz porównać wielkość jednego projektu z drugim, pominięcie komentarzy da lepszy obraz tego, z jaką ilością kodu masz faktycznie do czynienia.
- **Liczba logicznych wierszy kodu** (ang. *Logical Lines of Code*, LLOC) — w tym wskaźniku przyjęto, że każda instrukcja odpowiada jednemu wierszowi kodu. Zasadę jej działania ilustruje poniższy urywek kodu. Weźmy pod uwagę następujący wiersz:

  ```
  while($i < 5) { echo "test"; /* Zwiększ o jeden */ $i++; }
  ```

  Wskaźnik LOC ma tu wartość 1. Ponieważ są trzy instrukcje wykonywalne, wskaźnik LLOC wynosi 3, bo kod można zapisać również tak, by w jednym wierszu była tylko jedna instrukcja:

  ```
  while($i < 5) {
      echo "test";
      /* Zwiększ o jeden */
      $i++;
  }
  ```

  W powyższym przykładzie instrukcje wykonywalne wyróżniliśmy pogrubieniem. Komentarze, puste wiersze i takie elementy składni jak klamry nie są instrukcjami wykonywalnymi — właśnie dlatego całowierszowy komentarz i nawias klamrowy zamykający pętlę nie są liczone jako logiczne linie kodu.

Złożoność cyklomatyczna

Zamiast po prostu liczyć wiersze kodu, możemy też mierzyć jego złożoność — na przykład poprzez zliczanie **ścieżek wykonawczych** (ang. *execution paths*) wewnątrz funkcji. Najczęściej spotykanym służącym do tego wskaźnikiem jest **złożoność cyklomatyczna** (ang. *Cyclomatic Complexity*, CC). Została ona wprowadzona do użycia jeszcze pod koniec lat 70. ubiegłego wieku, mimo to wciąż jest przydatna. Pomysł kryjący się za tą zagadkową nazwą jest prosty: liczymy punkty decyzyjne, czyli instrukcje `if`, `while`, `for` i `case`. Dodatkowo jako jedną instrukcję liczymy również wejście do funkcji.

Działanie tego wskaźnika ilustruje poniższy przykład:

```
// pierwszy punkt decyzyjny
function someExample($a, $b)
{
    // drugi punkt decyzyjny
    if ($a < $b) {
        echo "1";
```

```
        } else {
            echo "2";
        }

        // trzeci punkt decyzyjny
        if ($a > $b) {
            echo "3";
        } else {
            echo "4";
        }
    }
```

Wskaźnik CC dla powyższego fragmentu kodu wynosi 3: wejście do funkcji liczy się jako jedna ścieżka decyzyjna, podobnie jak obie instrukcje `if`. Z kolei dwie instrukcje `else` zgodnie z definicją nie są brane pod uwagę, gdyż wchodzą w skład klauzul `if`. Wskaźnik ten przydaje się szczególnie do szybkiego szacowania złożoności nieznanego kodu. Często służy do sprawdzania pojedynczych funkcji, ale można go też stosować wobec klas, a nawet całych aplikacji. Jeśli funkcja ma wysoki wskaźnik CC, warto podzielić ją na kilka mniejszych, aby tę wartość obniżyć.

Złożoność NPath

Drugim wskaźnikiem złożoności kodu jest **złożoność NPath**. Koncepcja jest w zasadzie podobna, jak w przypadku CC, bo również liczone są ścieżki decyzyjne w funkcji. Zliczane są jednak *wszystkie* możliwe ścieżki, a nie tylko cztery instrukcje (`if`, `while`, `for` i `case`) jak we wskaźniku CC. Poza tym punkt wejścia do funkcji nie jest w tym przypadku liczony jako ścieżka decyzyjna.

Po przeanalizowaniu powyższego przykładu można stwierdzić, że złożoność NPath wynosi 4, bo mamy 2 · 2 możliwe ścieżki przejścia przez funkcję: obie instrukcje `if` oraz obie `else`. Wszystkie cztery instrukcje `echo` są więc uznawane za ścieżki decyzyjne. Jak wcześniej wspomniano, samo wywołanie funkcji nie jest brane pod uwagę. Gdybyśmy teraz dodali kolejną instrukcję `if`, złożoność NPath wzrosłaby do 8, dlatego że mielibyśmy wtedy 2 · 2 · 2 możliwe ścieżki. Innymi słowy, wskaźnik ten rośnie wykładniczo, więc szybko może osiągnąć dość dużą wartość.

Złożoność NPath, lepiej niż wskaźnik CC, obrazuje faktyczny nakład pracy konieczny do przetestowania funkcji, bo mówi bezpośrednio, ile rezultatów działania funkcji musimy sprawdzić w celu uzyskania stuprocentowego pokrycia testowego.

Wskaźniki Halsteada

Maurice Halstead, pod koniec lat 70. ubiegłego wieku, przedstawił zestaw ośmiu miar, które są stosowane do dzisiaj pod nazwą **wskaźników Halsteada**. Są one oparte wyłącznie na liczonych osobno sumach operatorów (np. `==`, `!=` i `&&`) i operandów (np. nazw

Kup ksi k Pole ksi k

funkcji, zmiennych i stałych), ale jak się przekonasz, to wystarczy, by uzyskać wiele informacji na temat badanego kodu.

Nie musimy dokładnie wiedzieć, jak obliczane są te wskaźniki. Jeśli Cię to interesuje, więcej informacji na ich temat uzyskasz pod adresem *https://www.verifysoft.com/en_halstead_metrics.html*. Warto jednak wiedzieć, że wskaźniki Halsteada to:

- **Długość** (ang. *length*) — policzenie łącznej sumy operatorów i operandów mówi nam, z jaką ilością kodu mamy do czynienia.
- **Zasób słownictwa** (ang. *vocabulary*) — samo podsumowanie liczb unikatowych operatorów i operandów pozwala już poznać złożoność kodu.
- **Wielkość** (ang. *volume*) — określa zawartość informacji w kodzie na podstawie długości i zasobu słownictwa.
- **Trudność** (ang. *difficulty*) — wskazuje podatność na błędy (to znaczy na ile prawdopodobne jest wprowadzenie w kodzie defektu).
- **Poziom** (ang. *level*) — jest odwrotnością trudności — słowem, im wyższy poziom kodu, tym łatwiej uniknąć błędów.
- **Nakład pracy** (ang. *effort*) — wysiłek umysłowy potrzebny do zrozumienia kodu.
- **Czas** (ang. *time*) — mówi nam, ile mniej więcej trwała implementacja.
- **Defekty** (ang. *bugs*) — to szacunkowa liczba błędów zawartych w kodzie.

Liczby te pozwolą Ci się z grubsza zorientować, z jakiego typu kodem masz do czynienia. Czy łatwo go zrozumieć? Jak długo go opracowywano? Ilu można spodziewać się w nim błędów? Jednak same te liczby, bez porównania ich z wynikami z innych aplikacji, niewiele Ci pomogą.

Indeks ryzyka zmian w kodzie

Innym bardzo przydatnym wskaźnikiem jest **indeks ryzyka zmian w kodzie** (ang. *Change Risk Anti-Patterns*, CRAP). Wykorzystywany jest w nim wskaźnik CC i pokrycie testami rozpatrywanego kodu.

Pokrycie kodu

Prawdopodobnie wiele razy obił Ci się o uszy termin **pokrycie kodu** (ang. *code coverage*). Jest to wskaźnik stosowany w kontekście testów automatycznych, opisujący liczbę wierszy kodu (wyrażoną jako odsetek całkowitej liczby wierszy), dla której zostały napisane testy jednostkowe. Wskaźnik ten i wymagania wstępne do jego policzenia omówimy szerzej w dalszej części książki, gdy zajmiemy się dokładniej tym zagadnieniem.

Kup ksi k Pole ksi k

Połączenie tych dwóch wskaźników jest dość przydatne. Kod, który nie jest przesadnie złożony i ma wysokie pokrycie testami, z dużo większym prawdopodobieństwem będzie mieć mniej błędów i łatwiej go będzie pielęgnować niż kod skomplikowany i niemający wielu testów.

Indeks utrzymywalności kodu

Ostatnim wskaźnikiem, któremu przyjrzymy się w tym punkcie jest **indeks utrzymywalności kodu** (ang. *maintainability index*). Jest to pojedyncza liczba określająca konserwowalność, inaczej mówiąc, stopień trudności jego zmiany bez wprowadzania nowych błędów. Wskaźnik ten jest dla nas szczególnie interesujący z dwóch powodów.

Po pierwsze, jest oparty na wcześniej wspomnianych wskaźnikach; do wyliczenia tego indeksu stosuje się LOC, wskaźniki Halsteada i CC. Znowu jednak: nie musimy faktycznie znać dokładnej formuły. Jeśli Cię to interesuje, możesz zajrzeć na stronę *https://www.verifysoft.com/en_maintainability.html*.

Po drugie, wartość tego wskaźnika pozwala bezpośrednio ocenić jakość kodu:

- 85 i powyżej — łatwy w utrzymaniu;
- od 65 do 85 — średnio trudny w utrzymaniu;
- 65 i poniżej — trudny w utrzymaniu.

W przypadku tej metryki nie potrzebujemy innego kodu do porównań. Dlatego właśnie jest szczególnie przydatna do szybkiej oceny jakości kodu.

W tym podrozdziale „zaliczyliśmy” mnóstwo teorii. Jak dotąd, idzie nam doskonale — na pewno nie będziesz żałować czasu poświęconego na naukę, bo za chwilę pokażemy, jak zbierać te wskaźniki przy użyciu jeszcze innych narzędzi dla języka PHP.

Zbieranie wskaźników w PHP

W tym podrozdziale przyjrzymy się narzędziom ze świata PHP do zbierania wskaźników jakości kodu. Jak wkrótce zobaczysz, wskaźniki te nie są tylko liczbami — pozwalają racjonalnie prognozować na temat nakładu pracy potrzebnej do zrefaktoryzowania kodu. Pomagają też zidentyfikować te fragmenty kodu, które wymagają szczególnej uwagi.

Po raz kolejny wyselekcjonowaliśmy narzędzia:

- *phploc*,
- PHP Depend,
- PhpMetrics.

Kup ksi k Pole ksi k

phploc

Jak dowiedzieliśmy się w poprzednim podrozdziale, akronim LOC pochodzi od angielskiego wyrażenia *lines of code* (liczba wierszy kodu), więc już sama nazwa wskazuje główne przeznaczenie tego narzędzia. Wskaźnik ten, choć podstawowy, mówi już co nieco o bazie kodu. Narzędzie *phploc* podaje też inne wskaźniki, na przykład CC, więc warto przyjrzeć się mu bliżej.

Instalacja i użytkowanie

Autor tego narzędzia, Sebastian Bergmann, zasłynął jako twórca programu *phpunit*, de facto standardu w dziedzinie testów automatycznych w świecie PHP. Zaleca, aby nie instalować narzędzia *phploc* za pomocą menedżera Composer, ale korzystać bezpośrednio z pliku *phar*. Zalety i wady takiego podejścia omówimy w następnym rozdziale. Na razie posłuchajmy po prostu rady autora i pobierzmy bezpośrednio archiwum *phar*:

```
$ wget https://phar.phpunit.de/phploc.phar
```

Polecenie to pobierze ostatnią wersję narzędzia *phploc* do bieżącego katalogu. Potem możemy za jego pomocą bezpośrednio przeskanować projekt:

```
$ php phploc.phar src
```

Skanowanie pojedynczych plików

Choć narzędzie *phploc* jest przeznaczone do skanowania całych projektów, można też podać pojedynczy plik. Choć mierzenie wartości średnich nie ma wtedy sensu, bo stosuje się je w odniesieniu do całego projektu; narzędzie to jest mimo wszystko przydatne, ponieważ pozwala znaleźć wskaźniki LOC lub CC dla klasy.

Powyższe polecenie przeskanuje folder *src* ze wszystkimi podfolderami, zbierze o nich informacje i przedstawi je bezpośrednio w wierszu poleceń, co widać na rysunku 8.1.

Zawarto tu dużo więcej informacji niż tylko sam wskaźnik LOC. Dane wyjściowe są podzielone na następujące kategorie:

- *Size* (wielkość) — oczywiście głównym powodem, dla którego to narzędzie istnieje, jest mierzenie wielkości projektu poprzez zliczanie wierszy kodu — na kilka sposobów przedstawionych w poprzednim podrozdziale. Nacisk położono na wskaźnik LLOC; dostępne są jego średnie wartości dla klas, metod klas i funkcji.
- *CC* — *phploc* — oblicza średnie wartości wskaźnika CC dla logicznych wierszy kodu, klas i metod.

```
$ php phploc.phar src
phploc 7.0.2 by Sebastian Bergmann.

Directories                                   8
Files                                        47

Size
  Lines of Code (LOC)                      3349
  Comment Lines of Code (CLOC)              199 (5.94%)
  Non-Comment Lines of Code (NCLOC)        3150 (94.06%)
  Logical Lines of Code (LLOC)              897 (26.78%)
    Classes                                 868 (96.77%)
      Average Class Length                   18
        Minimum Class Length                  0
        Maximum Class Length                117
      Average Method Length                   3
        Minimum Method Length                 0
        Maximum Method Length                40
      Average Methods Per Class               3
        Minimum Methods Per Class             1
        Maximum Methods Per Class            23
    Functions                                29 (3.23%)
      Average Function Length                 1
    Not in classes or functions               0 (0.00%)

Cyclomatic Complexity
  Average Complexity per LLOC              0.22
  Average Complexity per Class             4.89
    Minimum Class Complexity               1.00
    Maximum Class Complexity              51.00
  Average Complexity per Method            2.10
    Minimum Method Complexity              1.00
    Maximum Method Complexity             26.00

Dependencies
  Global Accesses                             0
    Global Constants                          0 (0.00%)
    Global Variables                          0 (0.00%)
    Super-Global Variables                    0 (0.00%)
  Attribute Accesses                        805
    Non-Static                              805 (100.00%)
    Static                                    0 (0.00%)
  Method Calls                              364
    Non-Static                              349 (95.88%)
    Static                                   15 (4.12%)
```

Rysunek 8.1. Przykładowe dane wyjściowe polecenia phploc (fragment)

- *Dependencies* (zależności) — sekcja ta mówi o tym, ile razy miał miejsce dostęp do stanu globalnego oraz statyczny dostęp do atrybutów i metod. Obie te operacje uznaje się za złe praktyki i należy ich unikać, więc liczby te dostarczają więcej wskazówek na temat jakości kodu.
- *Structure* (struktura) — w ostatniej sekcji danych wyjściowych (tej, która nie zmieściła się na rysunku 8.1) *phploc* podaje więcej szczegółów na temat struktury kodu. Nie ma wyraźnych reguł co do ich interpretacji; można jednak na ich podstawie wyciągnąć pewne wnioski. Przyjrzyjmy się na przykład następującym danym:

- Jak dużo jest przestrzeni nazw w stosunku do całej wielkości kodu? Obszerna baza kodu zawierająca niewiele przestrzeni nazw wskazuje, że projekt nie ma dobrej struktury.
- Czy są stosowane interfejsy i ile ich jest w stosunku do rozmiaru projektu? Korzystanie z interfejsów poprawia wymienność klas i wskazuje, że kod ma właściwą strukturę.

Tyle informacji o funkcjach narzędzia *phploc* na razie nam wystarczy. Ten prosty w użyciu, ale przydatny program pozwala szybko zorientować się w ogólnej jakości kodu oraz strukturze projektu, powinien więc stać się częścią Twojego zestawu narzędzi. Nie podaje on jednak interpretacji liczb — zrozumienie ich sensu wymaga pewnego doświadczenia.

PHP Depend

Gdyby przyznawano nagrodę za połączenie największej ilości wskaźników w jednym narzędziu, z pewnością trafiłaby ona do programu **PHP Depend** (PDepend). Podaje on wszystkie wskaźniki omówione w poprzednim podrozdziale i jeszcze wiele innych. Nie jest jednak najłatwiejszy w obsłudze, a informacjom w witrynie i w repozytorium daleko do ideału. Pomimo wszystko warto go jednak wypróbować.

Instalacja i użytkowanie

Narzędzie to, podobnie jak poprzednie, można zainstalować przy użyciu menedżera Composer albo pobrać bezpośrednio plik *phar*. Na razie spróbujemy tego pierwszego wariantu:

```
$ composer require pdepend/pdepend --dev
```

Jeśli nie było żadnych nieprzyjemnych niespodzianek, PDepend można uruchomić bezpośrednio:

```
$ vendor/bin/pdepend --summary-xml=pdepend_summary.xml src
```

Od razu widać, że PDepend wywodzi się z narzędzia jakości kodu Java o nazwie JDepend, ponieważ dane wyjściowe zapisywane są do pliku XML. Nazwę pliku podaje się po opcji `--summary-xml`. Jako argument musimy też podać folder do przeskanowania.

PDepend, jak widać z przykładowych danych wyjściowych, wyświetla kilka liczb:

```
PDepend 2.10.3

Parsing source files:
...............................................          47

Calculating Cyclomatic Complexity metrics:
.................                                        355
```

Kup książkę Poleć książkę

```
Calculating Node Loc metrics:
.............                                           279

Calculating NPath Complexity metrics:
.................                                       355

Calculating Inheritance metrics:
.....                                                   101
```

Pominęliśmy tutaj kilka wierszy. Liczby te mówią tylko, ile razy w podanym folderze były liczone poszczególne wskaźniki, więc bezpośrednie dane wyjściowe nie są specjalnie przydatne. Aby zobaczyć rzeczywiste wskaźniki, musimy otworzyć raport XML. W naszym przypadku wygenerowany plik nosi nazwę *pdepend_summary.xml*.

Raport XML jest zbyt wielki, by zamieścić go w tej książce, więc aby zobaczyć go w całej okazałości, najlepiej wypróbuj narzędzie samodzielnie. Możemy jednak przedstawić tu strukturę raportu:

```
<?xml version="1.0" encoding="UTF-8"?>
<metrics>
  <files>
    <file name="/ścieżka/do/przestrzeni/nazw/NazwaKlasy.php"/>
    <!-- ... -->
  </files>
  <package name="PrzestrzenNazw">
    <class name="NazwaKlasy" fqname="PrzestrzenNazw\NazwaKlasy">
      <file name="/ścieżka/do/przestrzeni/nazw/NazwaKlasy.php"/>
      <method name="nazwaMetody"/>
      <!-- ... -->
    </class>
    <!-- ... -->
  </package>
</metrics>
```

Węzeł `<metrics>` reprezentuje katalog, który został w całości przeskanowany. Ma on następujące **dzieci** (ang. *child nodes*):

- `<files>`, w którego dzieciach `<file>` wyszczególnione są wszystkie przeskanowane pliki.
- `<package>`, w którym wyszczególnione są poszczególne przestrzenie nazw. Wewnątrz niego znajdują się węzły `<class>`. Każda klasa ma zestaw węzłów `<method>`, po jednym dla każdej metody w klasie. Wreszcie — w kolejnym węźle `<file>` podana jest nazwa pliku zawierającego klasę.

Oczywiście nie są to wszystkie dane wyjściowe generowane przez narzędzie PDepend. Do każdego węzła dodawane są dziesiątki atrybutów, zawierających nazwy i wartości obliczonych wskaźników. Oto przykładowy węzeł z raportu XML, wygenerowanego ze źródeł samego narzędzia PDepend:

Kup ksi k Pole ksi k

```
<method name="setConfigurationFile" start="80" end="89"
  ccn="2" ccn2="2" loc="10" cloc="0" eloc="8" lloc="3"
  ncloc="10" npath="2" hnt="15" hnd="21"
  hv="65.884761341681" hd="7.3125" hl="0.13675213675214"
  he="481.78231731105" ht="26.765684295058"
  hb="0.020485472371812" hi="9.0098818928795"
  mi="67.295865328327"/>
```

Prawdopodobnie jesteś już w stanie rozpoznać niektóre wskaźniki, takie jak `lloc` (LOC) czy `ccn` (CC Number, wartość złożoności cyklomatycznej). Co do innych, wyjaśnienia (albo przynajmniej pełne nazwy skrótów występujących w raporcie XML) znajdziesz w dokumentacji internetowej *https://pdepend.org/documentation/software-metrics/index.html*.

Pozostałe opcje

PDepend ma dwie opcje, o których warto wiedzieć:

- `--exclude` — wyklucza ze skanowania przestrzeń nazw (w stosowanej tu terminologii: pakiet). Można podać wiele przestrzeni nazw, rozdzielonych przecinkami. Koniecznie ujmij je w cudzysłów:

  ```
  $ vendor/bin/pdepend --summary-xml=pdepend_summary.xml --
  exclude="Twoja\Przestrzen\Nazw,Inna\Przestrzen\Nazw" src
  ```

- `--ignore` — pozwala ignorować jeden lub więcej folderów. I znowu, nie zapomnij o cudzysłowach:

  ```
  $ vendor/bin/pdepend --summary-xml=pdepend_summary.xml --
  ignore="ścieżka/do/folderu,ścieżka/do/innego/folderu" src
  ```

Narzędzie to potrafi również generować obrazy w formacie SVG z dalszymi informacjami. Jednak w naszej książce nie opiszemy tej funkcji, bo istnieje lepsze narzędzie, które poznasz w następnym punkcie.

PDepend jest programem zaawansowanym, ale jednocześnie trudnym do ogarnięcia. Wygenerowane dane wyjściowe są mało czytelne i gdy tylko projekt stanie się nieco większy, robią się bezużyteczne — chyba że plik XML zostanie przetworzony przy użyciu innych narzędzi. Być może jednak pewnego dnia potrzebne Ci będą zaawansowane wskaźniki dostępne w tym programie albo przyjdzie Ci pracować nad projektem, w którym będzie on stosowany, więc przygotowaliśmy Cię na taką okoliczność.

PhpMetrics

Aż do tej pory w świecie wskaźników jakości kodu PHP mieliśmy do czynienia tylko z narzędziami tekstowymi. Teraz to się zmieni, bo przyjrzymy się programowi **PhpMetrics**, który generuje raporty dużo bardziej wpadające w oko, a nawet interaktywne.

Instalacja i użytkowanie

Dodajmy do projektu — przy użyciu menedżera Composer — narzędzie PhpMetrics:

```
$ composer require phpmetrics/phpmetrics --dev
```

Po pobraniu wszystkich plików można natychmiast zacząć generować pierwszy raport:

```
$ vendor/bin/phpmetrics --report-html=phpmetrics_report src
```

Opcja `--report-html` określa folder, w którym raport zostanie utworzony. Przeskanować można kilka folderów, podanych w postaci listy rozdzielonej przecinkami. W naszym przykładzie zbadamy jednak tylko folder *src*.

W rezultacie PhpMetrics pokaże kilka danych statystycznych, które już nam coś powiedzą na temat kodu. Rysunek 8.2 przedstawia fragment danych wyjściowych, które przypominają te generowane przez *phploc*.

```
Executing system analyzes...


Executing composer analyzes, requesting https://packagist.org...


LOC


    Lines of code                             876
    Logical lines of code                     762
    Comment lines of code                     114
    Average volume                            172.78
    Average comment weight                    14.48
    Average intelligent content               14.48
    Logical lines of code by class            35
    Logical lines of code by method           11
Object oriented programming
    Classes                                   22
    Interface                                 14
    Methods                                   69
    Methods by class                          3.14
    Lack of cohesion of methods               0.86

Coupling
    Average afferent coupling                 1.09
    Average efferent coupling                 2.95
    Average instability                       0.8
    Depth of Inheritance Tree                 1.47

Package
    Packages                                  7
    Average classes per package               5.14
    Average distance                          0.18
```

Rysunek 8.2. Dane wyjściowe konsoli z narzędzia PhpMetrics (fragment)

Kup książkę Poleć książkę

Aby zobaczyć dopiero co wygenerowany faktyczny raport HTML, po prostu otwórz w przeglądarce znajdujący się we wspomnianym wyżej folderze plik *index.html*. Zanim przyjrzymy się bliżej wygenerowanemu raportowi, zobaczmy najpierw, jakie inne przydatne opcje oferuje PhpMetrics:

- `--metrics` — opcja ta zwraca listę dostępnych wskaźników. Pozwala odszyfrować akronimy takie jak `mIwoC`.
- `--exclude` — w opcji tej można podać jeden lub więcej katalogów, które mają być wykluczone ze skanowania.
- `--report-[csv|json|summary-json|violations]` — pozwala zapisywać wynikowe raporty w formatach innych niż HTML, na przykład `--report-json`.

Otwieranie przeglądarki z wiersza poleceń

Jeśli używasz systemu operacyjnego opartego na jądrze Linux, na przykład Ubuntu, możesz szybko otworzyć plik HTML z poziomu wiersza poleceń w następujący sposób:

```
$ firefox phpmetrics_report/index.html
```

Ewentualnie tak:

```
$ chromium phpmetrics_report/index.html
```

Zapoznanie z raportem

Jeśli otwierasz raport narzędzia PhpMetrics po raz pierwszy, znajdziesz w nim wiele informacji. Nie będziemy wnikali w każdy drobny szczegół, ale pokażemy, które części raportu będą na początek, naszym zdaniem, najbardziej wartościowe.

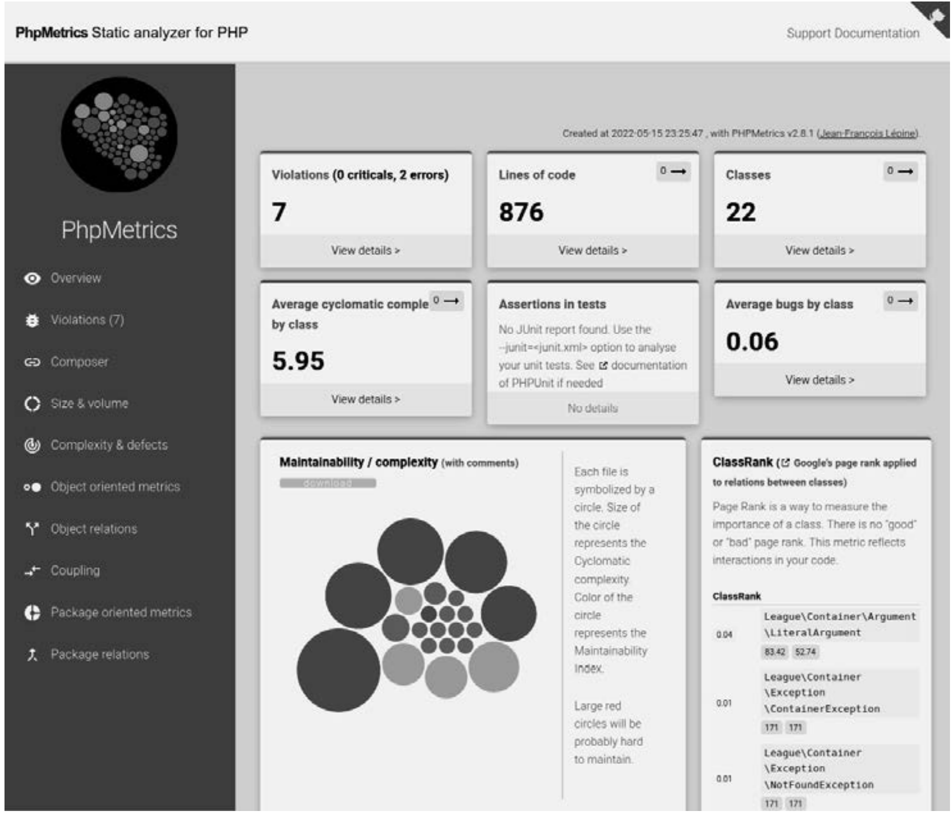
Aby lepiej zilustrować korzystanie z narzędzia PhpMetrics, jako bazę kodu (na której będziemy pracować) wybraliśmy w sposób losowy istniejący pakiet o otwartym kodzie źródłowym, o nazwie `thephpleague/container`. Jest to doskonały, zgodny z normą PSR-11, kontener wstrzykiwania zależności; o na tyle odpowiedniej wielkości kodu, że może posłużyć za przykład. Rysunek 8.3 przedstawia główną stronę przykładowego raportu, który wygenerowaliśmy.

Kluczowe wskaźniki

Po lewej stronie znajduje się menu, z którego można przejść do innych stron raportu. Górną część strony zajmuje kilka kluczowych wskaźników, z których najbardziej interesujące to:

- *Lines of code* (liczba wierszy kodu) — mówi więcej o wielkości tego projektu. Po kliknięciu tej etykiety przeniesiesz się na inną stronę, zawierającą szczegółową listę wszystkich klas i właściwych im wskaźników wielkości, takich jak LOC.

Kup książkę Poleć książkę



Rysunek 8.3. Główna strona raportu z narzędzia PhpMetrics

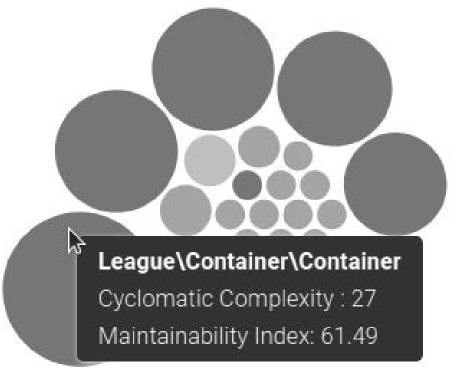
- *Violations* (naruszenia) — podaje liczbę wykrytych przez narzędzie PhpMetrics naruszeń norm. I znowu, po kliknięciu tej etykiety przeniesiesz się na inną stronę z listą klas i występujących w nich naruszeń — na przykład tego, czy są zbyt skomplikowane (*Too complex method code*, zbyt złożony kod metody), z dużym prawdopodobieństwem mają błędy (*Probably bugged*, prawdopodobnie wadliwe) albo czy korzystają ze zbyt wielu innych klas lub zależności (*Too dependent*, zbyt zależne).
- *Average cyclomatic complexity by class* (średnia złożoność cyklomatyczna klas) — informuje dokładnie o tym, co jest napisane na etykiecie. Widok szczegółów pozwala uzyskać więcej informacji o złożoności na poziomie klas.

Inne ramki również zawierają interesujące informacje, ale te wyżej wymienione z powodzeniem wystarczą do tego, by uzyskać szybki ogląd najbardziej problematycznych fragmentów kodu.

Utrzymywalność i złożoność

Pod kluczowymi wskaźnikami w raporcie PhpMetrics znajduje się między innymi diagram, który na pewno przykuł Twój wzrok już przy pierwszym otwarciu raportu: wykres *Maintainability/complexity* (utrzymywalność/złożoność). Składa się on z kolorowych kół obrazujących poszczególne przestrzenie nazw projektu. Rozmiar koła reprezentuje wskaźnik CC klasy. Im większe koło, tym większa złożoność. Kolor przedstawia indeks utrzymywalności, w zakresie od zielonego (wysoki) do czerwonego (niski).

Jeśli umieścisz kursor myszy na kole, zobaczysz, jaką przestrzeń nazw ono reprezentuje i dokładne wartości tych dwóch wskaźników (rysunek 8.4).



Rysunek 8.4. Wykres utrzymywalności i złożoności z wyskakującym okienkiem

Wykres ten jest bardzo przydatny, bo pozwala szybko uchwycić ogólną jakość kodu — im mniej dużych, czerwonych kół, tym lepiej. Dzięki temu łatwo dostrzec problematyczne składowe.

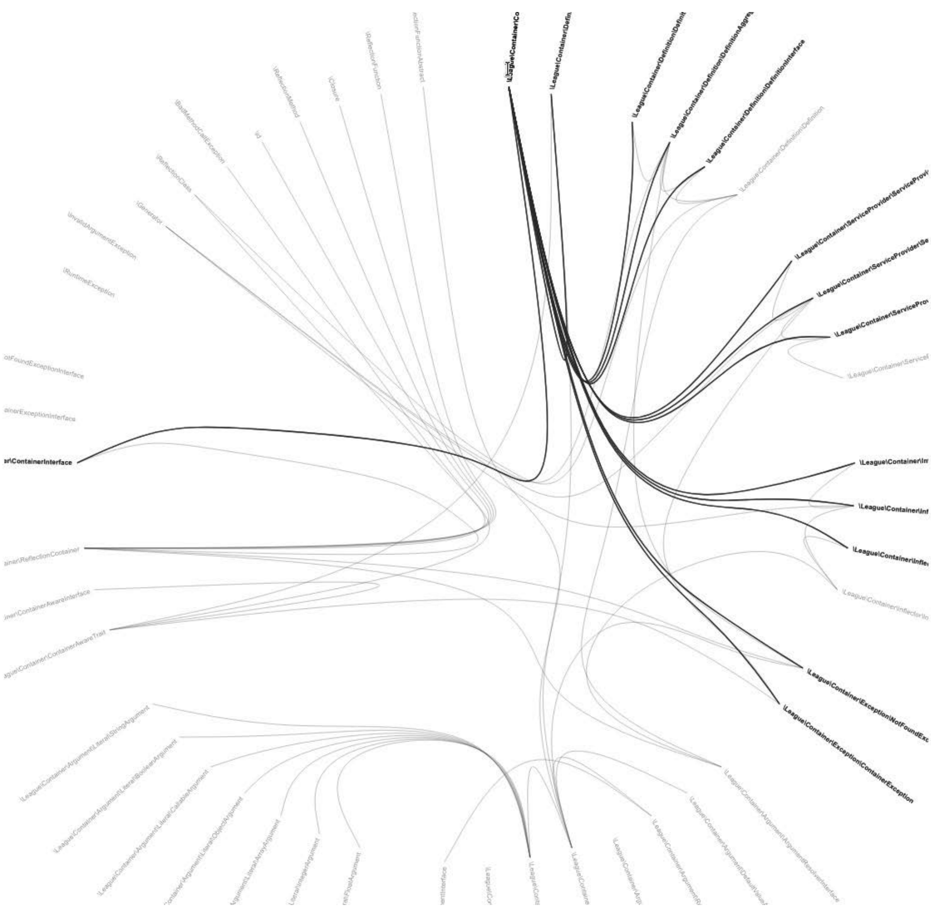
Relacje między obiektami

Gdy wybierzesz z menu po lewej stronie pozycję *Object relations* (relacje obiektów), pokaże się wykres przedstawiający relacje między poszczególnymi przestrzeniami nazw. Relacje danej przestrzeni zostaną wyróżnione po umieszczeniu kursora myszy na etykiecie tekstowej. Wykres jest duży, więc nie możemy go w tej książce przedstawić w całej okazałości, ale zobaczmy przynajmniej, jak na pierwszy rzut oka wygląda (rysunek 8.5).

Sprzężenie

Sprzężenie klas mówi o ich wzajemnych zależnościach. Są tu dwa główne wskaźniki:

- **Sprzężenia dośrodkowe** (ang. *afferent couplings*, *Ca*) — liczba klas zależnych od danej klasy. Jeśli jest ich wiele, oznacza to, że klasa jest w projekcie bardzo ważna.

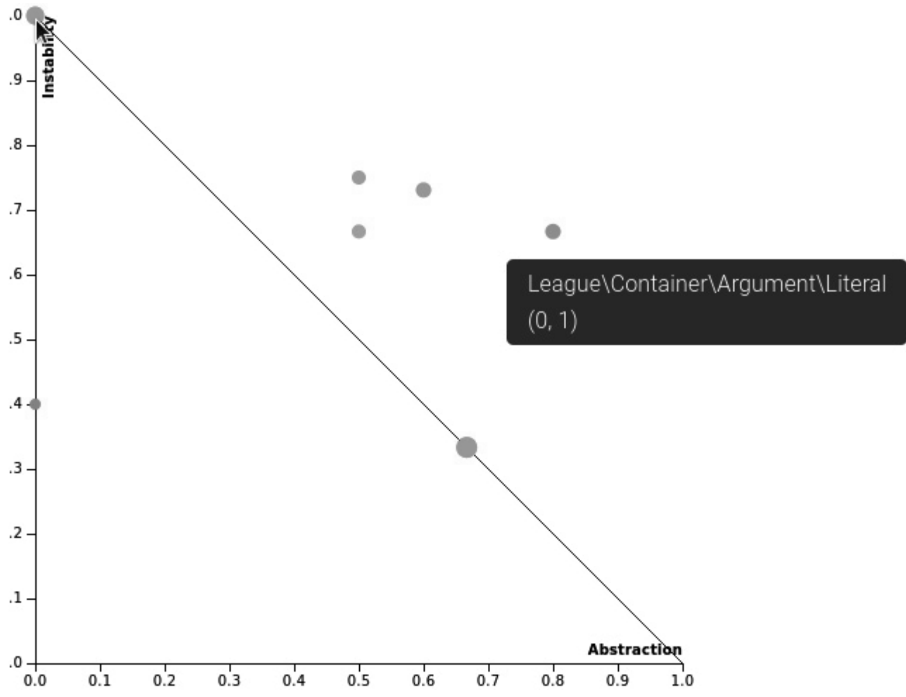


Rysunek 8.5. Wykres relacji między obiektami

- **Sprzężenia odśrodkowe** (ang. *efferent couplings*, *Ce)* — ich liczba informuje o tym, ile dana klasa ma zależności. Im jest ich więcej, tym bardziej klasa zależy od innych.

Wskaźniki dotyczące pakietów

Ostatni wykres, który pokażemy, nosi nazwę *Abstractness vs. Instability* (abstrakcyjność a niestabilność). Jak sama nazwa wskazuje, przedstawia on związek pomiędzy abstrakcyjnością i niestabilnością pakietów. Wprowadził go Robert Martin, jest oparty na jego pracy na temat wskaźników obiektowych. Przykładowy wykres widać na rysunku 8.6.



Rysunek 8.6. Wykres zależności między abstrakcyjnością i niestabilnością

Co jednak dokładnie oznaczają te dwa pojęcia w kontekście wytwarzania oprogramowania? Przyjrzyjmy się ich definicjom:

- **Abstrakcyjność** (ang. *abstractness*, *A*) — jest to stosunek liczby abstrakcyjnych klas bazowych i interfejsów do całkowitej liczby klas w przestrzeni nazw czy pakiecie. Im więcej w pakiecie typów abstrakcyjnych, tym łatwiejsze i mniej ryzykowne staje się wprowadzanie zmian. Wskaźnik *A* przyjmuje wartości od `0` (tylko klasy konkretne) do `1` (tylko typy abstrakcyjne).
- **Niestabilność** (ang. *Instability*, *I*) — mówi o wrażliwości pakietu na zmiany i wyraża się stosunkiem wartości wskaźnika *Ce* do sumy wartości *Ce* i *Ca* (*Ce* + *Ca*). Innymi słowy, im więcej zależności ma pakiet, tym mniej będzie stabilny. Wskaźnik *I* przyjmuje wartości od `0` (pakiet stabilny) do `1` (niestabilny).

Martin orzekł, że pakiety, które są stabilne, a więc w dużym stopniu niezależne od innych, powinny mieć też wskaźnik *A* na wysokim poziomie. I odwrotnie, pakiety niestabilne powinny składać się z konkretnych klas. Zatem zgodnie z teorią abstrakcyjność klasy (*A*) powinna równoważyć jej niestabilność (*I*). Oznacza to, że w idealnym przypadku suma *A* plus *I* powinna wynosić `1` ($A + I = 1$). To równanie obrazuje skośna linia od lewego górnego do prawego dolnego narożnika wykresu. Należy dążyć do tego, by pakiety znajdowały się blisko tej linii.

W prawdziwym raporcie pod wykresem znajduje się tabela z bardziej dokładnymi wartościami. Jeśli umieścisz wskaźnik myszy na kółku, pojawi się wyskakujące okienko z nazwą reprezentowanej przez nie klasy, a także wartościami wskaźników *A* (pierwsza liczba) oraz *I* (druga liczba).

Inne informacje

Na tym kończy się nasza prezentacja narzędzia PhpMetrics. Do zbadania pozostało jeszcze wiele, między innymi wskaźnik ClassRank, w którym klasy są szeregowane według ważności (to znaczy liczby interakcji z innymi elementami kodu), zgodnie z algorytmem PageRank firmy Google. Nie możemy w tej książce opisać wszystkiego — znasz już jednak wiele z tych wskaźników. Dość pomocna będzie dokumentacja narzędzia. Łącze do niej znajduje się w prawym górnym rogu każdej strony.

Zalety i wady korzystania ze wskaźników

Dwa ostatnie z dotychczasowych rozdziałów tej książki pozwoliły Ci poznać wiele narzędzi i wskaźników, które istnieją tylko po to, by pomagać w pisaniu lepszego oprogramowania. W ciągu kilku minut możesz dodać do swojego projektu wiedzę, mądrość i wielogodzinną pracę setek (jeśli nie tysięcy) inżynierów oprogramowania.

Być może jednak przytłoczyła Cię już zupełnie liczba dostępnych możliwości. Które narzędzia najlepiej wybrać? Na których wskaźnikach należy w przyszłości się skupić?

Jeśli masz takie odczucia, nie przejmuj się. Nie porzucimy Cię w tym bałaganie — w następnych rozdziałach pomożemy znaleźć konfigurację, która będzie odpowiadała Twoim potrzebom. Na początek poświęćmy trochę czasu, aby przyjrzeć się zaletom, ale też i wadom korzystania ze wskaźników jakości kodu.

Zalety

Po pierwsze, każdy projekt informatyczny jest dziełem unikatowym. Jego rozwój opiera się na pewnych uwarunkowaniach, takich jak umiejętności programistów i dostępne w danym czasie szkielety aplikacyjne, ale podlega również czynnikom zewnętrznym, na przykład terminom realizacji, które dość często negatywnie wpływają na jakość kodu.

Po drugie, wskaźniki kodu pozwalają uzyskać ogólny ogląd bieżącego stanu projektu. Jeśli na przykład przejmujesz projekt stworzony przez byłego członka zespołu, z pewnością zechcesz się dowiedzieć, co Cię czeka. Zorientowawszy się w jakości kodu, możesz od razu skorygować swoje szacunki dotyczące nakładu pracy niezbędnego do realizacji przyszłych zleceń, niezależnie od tego w jakim kierunku pójdą.

Kup ksi k Pole ksi k

Wskaźniki jakości kodu pozwalają też dowiedzieć się, w którym miejscu musi on zostać ulepszony. Refaktoryzacja kodu jest doskonałym treningiem, a dzięki korzystaniu ze wskaźników będziesz wiedzieć, czy się ona powiodła. Niezależnie od tego, czy pracujesz nad własnym, hobbystycznym projektem, czy w zespole, a może chcesz też wnieść wkład do projektu o otwartym kodzie źródłowym; uzyskanie w raportach większej liczby zielonych lampek jest zawsze miłym osiągnięciem.

Jeśli natrafisz na fragment kodu, który z uzasadnionych względów pilnie wymaga refaktoryzacji, a kierownik projektu nie wyrazi na to zgody, możesz przy użyciu wskaźników wytłumaczyć, jak strasznie to wygląda i że nie jest to tylko Twoja subiektywna opinia. Wskaźniki kodu są bezstronne i (aż do bólu) uczciwe.

Ostatnią ważną funkcją tych wskaźników jest przede wszystkim to, że zapobiegają pisaniu złego kodu. Czasami pisanie kodu zgodnego z tymi wszystkimi regułami może być nieco nudne, ale z pewnością ten wysiłek w końcu się opłaci.

Wady

Powiedzieliśmy wcześniej, że terminy realizacji mogą zaszkodzić jakości kodu, bo nie pozwalają zrefaktoryzować zawartych w nim przykrych zapachów ani dodać większej liczby testów. Chociaż to prawda, musimy mieć świadomość, że niektórzy programiści, gdy przystąpią do mierzenia jakości swojego kodu, zaczynają go refaktoryzować w dużo większym stopniu, niż to konieczne, bo są nagradzani lepszymi wskaźnikami. Dlaczego jest to problem?

Wyobraź sobie na przykład, że masz w projekcie klasę o niskim indeksie utrzymywalności i wysokiej złożoności NPath, na którą wystarczy spojrzeć, a od razu widać, jak źle jest napisana. Z biegiem czasu osiągnęła jednak dojrzałość, a w pewnym okresie była poprawiana tyle razy, że okazało się, iż działa bezbłędnie. Narzędzia mówią jednak, że klasa ta ma niską jakość. Czy należy zacząć ją refaktoryzować?

Na takie pytanie nie da się oczywiście wyraźnie odpowiedzieć — ani twierdząco, ani przecząco. Jak wcześniej wspomniano, jeśli pracujesz nad kodem w wolnym czasie, refaktoryzacja klasy po to, by usunąć z niej większość przykrych zapachów ma sens (i sprawia przyjemność). Jeśli pracujesz nad komercyjnymi projektami i zarabiasz na życie programowaniem, nie zawsze będziesz mieć na to czas. Trzeba usuwać błędy, z powodu których użytkownicy oprogramowania są niezadowoleni, oraz implementować funkcje, na które niecierpliwie czekają. W końcu to właśnie dzięki usatysfakcjonowanym klientom płacisz swoje rachunki. Znalezienie złotego środka pomiędzy szybkością wytwarzania kodu a jego jakością nigdy nie jest łatwe — musisz po prostu mieć świadomość, że czasem trzeba przełknąć gorzką pigułkę i chwilowo zostawić zły kod w spokoju.

Kup ksi k Pole ksi k

Nie wykorzystuj wskaźników do konkurowania ze współpracownikami albo, co gorsza, wypowiadania się w zły sposób o programistach, którzy zostawili Cię sam na sam z projektem. Pamiętaj, że każdy pracuje na tyle dobrze, na ile pozwalają jego umiejętności. Nikt specjalnie nie stara się pisać złego kodu — zwykle zdarza się to dlatego, że programiści nigdy nie słyszeli o zasadach czystego kodu albo byli pod taką presją czasu, że musieli programować metodą „kopiuj-wklej", by zadowolić kierownictwo i klientów. Twoje środowisko pracy powinno być miejscem wzajemnego szacunku, uczynności i tolerancji, a nie konkurencji.

Podsumowanie

W tym rozdziale przedstawiliśmy kilka najczęściej stosowanych w świecie PHP wskaźników jakości kodu. Ponadto zaprezentowaliśmy narzędzia do ich zbierania. Istnieje oczywiście wiele innych wskaźników, nieopisanych w tej książce, ale nie musisz ich wszystkich znać — masz już solidną wiedzę, która pomoże Ci w codziennej pracy.

Narzędzia i wskaźniki jakości kodu na pewno nie są panaceum na wszystkie problemy. Z jednej strony bardzo pomagają w ulepszaniu kodu. Z drugiej, nie należy ich traktować jako najlepszego wyznacznika. Istnieje wiele przykładów udanego oprogramowania, które nigdy nie zaliczyłoby pomyślnie takiej kontroli jakości — chociażby WordPress. Gdyby jednak jego twórcy wiedzieli o tym wcześniej, z pewnością podeszliby do sprawy inaczej.

W następnym rozdziale pozostawimy za sobą sferę teorii. Nauczymy się, jak w ramach projektu uporządkować narzędzia przedstawione w ostatnich dwóch rozdziałach. Każdy projekt jest unikatowy, więc zaproponujemy różne warianty, które będziesz w stanie dopasować do swoich potrzeb.

Materiały dodatkowe

- *dePHPend* (*https://dephpend.com/*) to narzędzie, które potrafi rysować diagramy UML dla kodu PHP i pozwala wykrywać problemy z jego architekturą.

Kup ksi k Pole ksi k

PROGRAM PARTNERSKI

GRUPY HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

Zostań mistrzem czystego kodu!

PHP jest uważany za łatwy język programowania. Początkujący programiści dość szybko uruchamiają w nim swoje pierwsze aplikacje. Nie sprzyja to jednak wyrabianiu nawyków przestrzegania dobrych praktyk. Najczęściej adepci PHP tworzą niechlujny kod, który jest trudny w utrzymaniu, a ewentualne modyfikacje czy rozbudowa aplikacji sprawiają ogromne problemy. Kolejną kwestią jest brak czytelności kodu, co praktycznie uniemożliwia pracę zespołową nad projektem.

To książka adresowana do początkujących programistów, którzy chcą zacząć pisać czysty kod w PHP. Znajdziesz w niej przystępnie wyjaśnione koncepcje, a także szereg wskazówek, opisów dobrych praktyk, wzorców projektowych i technik pracy. Treść została podzielona na dwie części. W pierwszej wyjaśniono paradygmat pisania czystego kodu i zasady, które stosuje się w tym zakresie w języku PHP. W drugiej części omówiono najlepsze narzędzia, wzorce i praktyki, pokazano też, jak należy skonfigurować swoje zintegrowane środowisko programistyczne (IDE) i jak pozyskiwać wskaźniki mówiące o kondycji kodu. Sporo miejsca poświęcono testom automatycznym, pisaniu dokumentacji i innym istotnym zagadnieniom.

W książce:

- solidne podstawy pisania czystego kodu
- wskaźniki określające jakość kodu
- podstawy testów automatycznych
- implementacja ciągłej integracji w aplikacjach PHP
- wzorce projektowe, dobre praktyki i inne rozwiązania ułatwiające pracę

Carsten Windler od lat programuje w PHP. Kierował wieloma zespołami programistycznymi. Często występuje na konferencjach, pisze też artykuły do czasopism branżowych.

Alexandre Daubois jest liderem i współtwórcą Symfony, artystą cyfrowym, pisarzem i półmaratończykiem. Regularnie występuje na konferencjach branżowych.