

# MISTRZ PHP

PISZ NOWOCZESNY KOD

DAVEY SHAFIK  
LORNA MITCHELL  
MATTHEW TURLAND



WYKORZYSTAJ NAJNOWSZE TECHNIKI PROGRAMOWANIA,  
DZIĘKI KTÓRYM OSIĄGNIESZ WYŻSZY POZIOM ZAAWANSOWANIA

Tytuł oryginału: PHP Master: Write Cutting-edge Code

Tłumaczenie: Łukasz Piwko

ISBN: 978-83-246-4472-8

© Helion 2012.

Authorized Polish translation of the English edition of PHP Master, 1st Edition  
ISBN 9780987090874 © 2011, SitePoint Pty. Ltd.

This translation is published and sold by permission of O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

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 jakiegokolwiek 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 Wydawnictwo HELION 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 Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION  
ul. Kościuszki 1c, 44-100 GLIWICE  
tel. 32 231 22 19, 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/misphp>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

# Spis treści

---

Wstęp .....	13
Adresaci książki .....	13
Zawartość książki .....	14
Strona internetowa książki .....	16
Podziękowania .....	16
Konwencje typograficzne .....	17
Wskazówki, uwagi i ostrzeżenia .....	18
<b>Rozdział 1. Programowanie obiektowe .....</b>	<b>19</b>
Dlaczego programowanie obiektowe .....	19
Terminologia obiektowa .....	19
Wprowadzenie do programowania obiektowego .....	20
Deklarowanie klas .....	20
Konstruktory .....	21
Tworzenie obiektów .....	21
Automatyczne ładowanie .....	22
Używanie obiektów .....	23
Własności i metody statyczne .....	23
Obiekty i przestrzenie nazw .....	24
Dziedziczenie .....	27
Obiekty i funkcje .....	29
Określanie typów parametrów .....	29
Polimorfizm .....	29
Obiekty i referencje .....	30
Przekazywanie obiektów jako parametrów funkcji .....	31
Płynne interfejsy .....	32
Słowa kluczowe public, private i protected .....	33
Modyfikator public .....	33
Modyfikator private .....	33
Modyfikator protected .....	34
Wybór zakresu dostępności .....	34
Kontrola dostępności przy użyciu metod sprawdzających i ustawiających .....	35
Magiczne metody __get i __set .....	36
Interfejsy .....	37
Interfejs Countable z biblioteki SPL .....	37
Liczenie obiektów .....	37
Deklarowanie i używanie interfejsów .....	38
Identyfikowanie obiektów i interfejsów .....	39

Wyjątki .....	40
Obsługa wyjątków .....	40
Dlaczego należy używać wyjątków .....	41
Zgłaszanie wyjątków .....	41
Rozszerzanie klas wyjątków .....	41
Przechwytywanie wybranych typów wyjątków .....	42
Ustawianie globalnej procedury obsługi wyjątków .....	43
Wywołania zwrotne .....	44
Metody magiczne — zaawansowane wiadomości .....	44
Metody <code>__call()</code> i <code>__callStatic()</code> .....	45
Drukowanie zawartości obiektów przy użyciu metody <code>__toString()</code> .....	46
Serializacja obiektów .....	46
Osiągnięte cele .....	48
<b>Rozdział 2. Bazy danych .....</b>	<b>49</b>
Dane trwałe i aplikacje sieciowe .....	49
Sposoby składowania danych .....	50
Budowanie serwisu z przepisami na podstawie bazy MySQL .....	51
Tworzenie tabel .....	51
Rozszerzenie PDO .....	53
Łączenie się z bazą MySQL przy użyciu PDO .....	53
Pobieranie danych z tabel w bazie .....	54
Tryby pobierania danych .....	54
Parametry i instrukcje preparowane .....	55
Wiązanie wartości i zmiennych z instrukcjami preparowanymi .....	57
Wstawianie wiersza i pobieranie jego identyfikatora .....	58
Sprawdzanie liczby wstawionych, zmienionych i usuniętych rekordów .....	59
Usuwanie danych .....	60
Obsługa błędów w PDO .....	60
Obsługa błędów związanych z przygotowaniem zapytań .....	60
Obsługa błędów związanych z wykonywaniem zapytań .....	61
Obsługa błędów związanych z pobieraniem danych .....	62
Zaawansowane funkcje PDO .....	63
Transakcje a PDO .....	63
Procedury składowane i PDO .....	64
Projektowanie bazy danych .....	65
Klucze główne i indeksy .....	65
Polecenie MySQL Explain .....	65
Złączenia wewnętrzne .....	69
Złączenia zewnętrzne .....	70
Funkcje agregujące i grupowanie .....	71
Normalizacja danych .....	72
Podsumowanie .....	74

<b>Rozdział 3. Interfejsy programistyczne .....</b>	<b>75</b>
Zanim zaczniesz .....	75
Narzędzia do pracy z API .....	75
Dodawanie API do systemu .....	76
Architektura usługowa .....	76
Formaty danych .....	77
Format JSON .....	77
Format XML .....	79
HTTP — protokół przesyłania hipertekstu .....	82
Dane przesyłane w nagłówkach HTTP .....	82
Wysyłanie żądań HTTP .....	83
Kody statusu HTTP .....	87
Nagłówki HTTP .....	87
Czasowniki HTTP .....	91
Kryteria wyboru typów usług .....	92
PHP i SOAP .....	92
Opis usług SOAP za pomocą języka WSDL .....	94
Diagnozowanie HTTP .....	95
Gromadzenie informacji w dzienniku .....	95
Kontrola ruchu HTTP .....	96
Usługi RPC .....	96
Korzystanie z usług RPC: przykład na podstawie serwisu Flickr .....	97
Tworzenie usługi RPC .....	98
Usługi sieciowe a Ajax .....	100
Żądania międzydomenowe .....	104
Usługi RESTful .....	106
Więcej niż piękne adresy URL .....	107
Zasady usług RESTful .....	107
Budowanie usługi RESTful .....	108
Projektowanie usługi sieciowej .....	114
Do usług .....	115
<b>Rozdział 4. Wzorce projektowe .....</b>	<b>117</b>
Czym są wzorce projektowe .....	117
Wybieranie wzorca .....	117
Wzorzec singleton .....	118
Cechy .....	119
Wzorzec rejestr .....	120
Wzorzec fabryka .....	124
Wzorzec iterator .....	125
Wzorzec obserwator .....	133

Wzorzec wstrzykiwanie zależności .....	136
Wzorzec model-widok-kontroler .....	139
Tworzenie wzorców .....	150
<b>Rozdział 5. Bezpieczeństwo .....</b>	<b>151</b>
Działaj jak paranoik .....	151
Filtruj dane wejściowe, koduj dane wyjściowe .....	152
Filtrowanie i weryfikacja .....	152
Cross-site scripting .....	153
Atak .....	154
Obrona .....	155
Materiały w internecie .....	155
Cross-site Request Forgery .....	156
Atak .....	156
Obrona .....	157
Materiały w internecie .....	159
Session fixation .....	159
Atak .....	159
Obrona .....	160
Materiały w internecie .....	160
Session hijacking .....	161
Atak .....	161
Obrona .....	162
Materiały w internecie .....	163
SQL injection .....	163
Atak .....	163
Obrona .....	164
Materiały w internecie .....	165
Przechowywanie haseł .....	165
Atak .....	165
Obrona .....	166
Materiały w internecie .....	167
Atak siłowy .....	167
Atak .....	167
Obrona .....	169
Materiały w internecie .....	169
SSL .....	170
Atak .....	170
Obrona .....	171
Materiały w internecie .....	171
Dodatkowe zasoby .....	172

<b>Rozdział 6. Wydajność .....</b>	<b>173</b>
Benchmarking .....	173
Dostrajanie systemu .....	179
Zapisywanie kodu w pamięci podręcznej .....	179
Ustawienia inicjacyjne .....	184
Bazy danych .....	184
System plików .....	185
Buforowanie .....	185
Profilowanie .....	192
Instalowanie narzędzia XHProf .....	193
Instalowanie XHGui .....	197
Podsumowanie .....	204
<b>Rozdział 7. Automatykacja testów .....</b>	<b>205</b>
Testy jednostkowe .....	205
Instalowanie narzędzia PHPUnit .....	206
Pisanie przypadków testowych .....	206
Wykonywanie testów .....	208
Dublery .....	210
Pisanie kodu przystosowanego do testowania .....	213
Pisanie testów dla widoków i kontrolerów .....	217
Testowanie baz danych .....	221
Przypadki testowe baz danych .....	221
Połączenia .....	222
Zbiory danych .....	223
Asercje .....	225
Testowanie systemowe .....	226
Wstępna konfiguracja .....	226
Polecenia .....	227
Lokalizatory .....	228
Asercje .....	229
Integracja z bazą danych .....	230
Diagnozowanie usterek .....	231
Automatykacja pisania testów .....	232
Testowanie obciążeniowe .....	233
ab .....	233
Siege .....	234
Wypróbowane i przetestowane .....	236

<b>Rozdział 8. Kontrola jakości</b> .....	<b>237</b>
Pomiar jakości za pomocą narzędzi analizy statycznej .....	237
Narzędzie phploc .....	238
Narzędzie phpcpd .....	239
Narzędzie phpmd .....	240
Standardy kodowania .....	241
Weryfikacja kodu pod kątem standardów kodowania przy użyciu narzędzia PHP_CodeSniffer .....	241
Przeglądanie przypadków naruszenia reguł standardów kodowania .....	243
Standardy kodowania w narzędziu PHP_CodeSniffer .....	244
Dokumentacja i kod .....	244
Narzędzie phpDocumentor .....	246
Inne narzędzia dokumentacyjne .....	248
Kontrola źródła .....	248
Praca z centralnym systemem kontroli wersji .....	249
Kontrola źródła przy użyciu systemu Subversion .....	250
Projektowanie struktury repozytorium .....	252
Rozproszone systemy kontroli wersji .....	254
Społecznościowe narzędzia dla programistów .....	255
Kontrola kodu źródłowego przy użyciu narzędzia Git .....	255
Repozytorium jako centrum procesu budowy .....	257
Automatyzacja procesu wdrażania .....	257
Natychmiastowe przełączanie na nową wersję .....	257
Zarządzanie zmianami w bazie danych .....	258
Automatyzacja wdrażania i plik konfiguracyjny Phing .....	259
Gotowi do wdrażania .....	261
<b>Dodatek A Biblioteki PEAR i PECL</b> .....	<b>263</b>
Biblioteka PEAR .....	263
Biblioteka PECL .....	263
Instalowanie pakietów .....	264
Kanały PEAR .....	266
Używanie kodu PEAR .....	268
Instalowanie rozszerzeń .....	268
Ręczne kompilowanie rozszerzeń .....	269
Tworzenie pakietów .....	272
Kontrola wersji pakietów .....	276
Tworzenie kanału .....	277
Co dalej .....	280



<b>Dodatek B SPL: Standard PHP Library .....</b>	<b>281</b>
Interfejsy ArrayAccess i ArrayObject .....	281
Automatyczne wczytywanie .....	282
Praca na katalogach i plikach .....	283
Interfejs Countable .....	285
Struktury danych .....	286
Tablice o stałym rozmiarze .....	286
Listy .....	286
Stosy i kolejki .....	287
Stery .....	287
Kolejki priorytetowe .....	288
Funkcje .....	288
 <b>Dodatek C Dalszy rozwój .....</b>	 <b>289</b>
Czytaj, czytaj, czytaj .....	289
Uczestnictwo w wydarzeniach .....	290
Grupy użytkowników .....	291
Społeczności internetowe .....	291
Projekty typu open source .....	292
 <b>Skorowidz .....</b>	 <b>293</b>



# Rozdział 8

## Kontrola jakości

---

Ten rozdział jest kontynuacją opisanego w poprzednim rozdziale tematu automatyzacji testów. Poznasz w nim narzędzia pozwalające zapewnić wysoką jakość pisanych przez siebie programów. Wśród nich znajdują się programy do kontroli wersji kodu źródłowego, wspomagające współpracę między członkami zespołu programistycznego i ułatwiające panowanie nad rozwojem programu, oraz narzędzia do automatycznego wdrażania systemów do środowiska produkcyjnego, które w odróżnieniu od człowieka o niczym nie zapominają. Ponadto poznasz techniki analizy kodu w celu sprawdzenia, czy jest spójny i dobrze sformatowany, oraz dowiesz się, jak się generuje dokumentację z kodu źródłowego.

Wymienione narzędzia to elementy każdego dobrze prowadzonego projektu, w którym ograniczono do minimum ilość czasu potrzebnego na zajmowanie się mniej ważnymi aspektami technicznymi, a więcej pozostawiono na budowę wspianiałego programu.

## Pomiar jakości za pomocą narzędzi analizy statycznej

---

**Analiza statyczna** polega na badaniu kodu bez jego uruchamiania. Narzędzia służące do jej wykonywania oceniają kod w takiej postaci, w jakiej jest zapisany w plikach. Istnieje wiele programów, których można używać do tego celu, a co ciekawe, za najlepsze z nich nie trzeba płacić. Dzięki tym narzędziom można uzyskać ogólny obraz podstawy kodu (lub jej części), nawet gdy jest ona bardzo skomplikowana i obszerna.

Narzędzia do analizy statycznej stanowią jeden z kluczowych składników projektu programistycznego, ale są naprawdę przydatne tylko wtedy, kiedy włącza się je regularnie, najlepiej za każdym razem po zatwierdzeniu kodu w systemie zarządzania kodem źródłowym. Zwracają wiele cennych informacji na temat kodu (od liczby klas i wierszy, po wskazanie podobnych fragmentów), co może oznaczać, że zostały skopiowane z jednego miejsca i wklejone do innego! Pokażemy, w jaki sposób narzędzia do analizy statycznej pomagają zapanować nad dwoma niezmiernie ważnymi aspektami: standardami kodowania i dokumentacją.

Wszystkie narzędzia, które opisano w tej części rozdziału, można pobrać z biblioteki PEAR — opis, jak instaluje się programy przy użyciu tego narzędzia do zarządzania pakietami, znajduje się w dodatku A. Niektóre z tych programów mogą też być dostępne poprzez menedżer pakietów systemu operacyjnego, jeśli używasz jednego z systemów uniksowych. Możesz skorzystać z tej możliwości, ale pamiętaj, że istnieje duże ryzyko, iż wersje tak dostępnych aplikacji będą nieaktualne.

## Narzędzie phploc

Nazwa PHP Lines of Code (wiersze kodu PHP) może nie brzmi zbyt zachęcająco, ale narzędzie to dostarcza naprawdę cennych informacji, zwłaszcza gdy jest uruchamiane wielokrotnie przez pewien czas. Dzięki niemu można poznać topologię i rozmiar projektu. Oto wynik zwrócony przez narzędzie phploc dla standardowej wersji systemu WordPress:

```
$ phploc wordpress/
phploc 1.6.1 by Sebastian Bergmann.

Directories:                26
Files:                      380

Lines of Code (LOC):        171170
  Cyclomatic Complexity / Lines of Code:    0.19
Comment Lines of Code (CLOC):    53521
Non-Comment Lines of Code (NCLOC): 117649

Namespaces:                 0
Interfaces:                  0
Classes:                     190
  Abstract:                   0 (0.00%)
  Concrete:                   190 (100.00%)
  Average Class Length (NCLOC):    262
Methods:                     1990
  Scope:
    Non-Static:               1986 (99.80%)
    Static:                    4 (0.20%)
  Visibility:
    Public:                   1966 (98.79%)
    Non-Public:                24 (1.21%)
  Average Method Length (NCLOC):    25
  Cyclomatic Complexity / Number of Methods: 5.56

Anonymous Functions:        0
Functions:                   2330

Constants:                   351
  Global constants:          348
  Class constants:           3
```

System ten zawiera bardzo dużą liczbę wierszy kodu, a ponieważ jego początki sięgają dość daleko w przeszłość, elementów języka PHP 5 jest w nim niewiele. Dzięki phploc można sprawdzić rozmiar nieznanego programu albo analizować przyrost i zmiany kodu we własnym projekcie. Aby użyć phploc, należy zastosować następujące polecenie:

```
phploc wordpress/
```

Jego wynik będzie podobny do pokazanego powyżej. Dane te można zapisać w różnych formatach, np. XML do użytku w systemie ciągłej integracji.



### Złożoność cyklomatyczna

Złożoność cyklomatyczna to, najprościej mówiąc, miara określająca liczbę ścieżek wykonania funkcji (tzn. pokazująca, jak bardzo ta funkcja jest skomplikowana), związana z tym, ile testów może być potrzebnych do całkowitego przetestowania funkcji. Ogólnie rzecz biorąc: bardzo wysoka wartość oznacza, że lepiej przepisać funkcję od nowa i wydzielić z niej mniejsze metody, które będzie łatwiej przetestować.

## Narzędzie phpcpd

PHP Copy Paste Detector (wykrywacz kopiowania i wklejania kodu PHP) to narzędzie wyszukiujące w kodzie źródłowym takie same fragmenty w celu wykrycia części, które zostały skopiowane z jednego miejsca i wklejone do innego. Warto regularnie korzystać z tego dodatku, mimo że nie da się określić, jaki wynik jest najlepszy, jest to bowiem miara zależna od konkretnego projektu. W ramach przykładu ponownie użyjemy systemu WordPress, ponieważ jest to dobrze wszystkim znany projekt typu open source:

```
$ phpcpd wordpress/  
phpcpd 1.3.2 by Sebastian Bergmann.
```

```
Found 33 exact clones with 562 duplicated lines in 14 files:
```

- wp-admin/includes/update-core.php:482-500  
wp-admin/includes/file.php:733-751
- wp-admin/includes/class-wp-file-system-ssh2.php:346-365  
wp-admin/includes/class-wp-file-system-direct.php:326-345
- ...
- wp-includes/class-simplepie.php:10874-10886  
wp-includes/class-simplepie.php:13185-13197
- wp-content/plugins/akismet/admin.php:488-500  
wp-content/plugins/akismet/admin.php:537-549
- wp-content/plugins/akismet/legacy.php:234-248  
wp-content/plugins/akismet/legacy.php:301-315

```
0.33% duplicated lines out of 171170 total lines of code.
```

```
Time: 6 seconds, Memory: 154.50Mb
```

Dane takie szczególnie warto analizować przez pewien czas. To narzędzie może również zapisywać wyniki w formacie XML zrozumiałym dla systemu ciągłej integracji, dzięki czemu można je bezproblemowo dodać do skryptów wdrożeniowych i zwrócone informacje przedstawić w postaci wykresu. Mając wykaz nowych fragmentów kodu, które są do siebie podobne, można wykryć wszystkie duplikaty kodu i zastanowić się nad możliwościami wielokrotnego wykorzystania kodu. Należy jednak pamiętać, że nie zawsze ponowne użycie kodu jest korzystne. Zawsze warto rozważyć taką możliwość, jednak nie należy z tym przesadzać.

## Narzędzie phpmnd

PHP Project Mess Detector (wykrywacz bałaganu w PHP) to narzędzie pozwalające znaleźć w kodzie tzw. „śmierdzące fragmenty” (ang. *code smells*). Przeszukuje ono kod, stosując szereg metryk, aby znaleźć fragmenty, które wydają się nie w porządku. Program ten zwraca bardzo dużo danych, ale większość z nich to tylko dobre rady. Poniżej znajduje się fragment wyniku wyszukiwania problemów związanych z nazwami w systemie WordPress:

```
$ phpmnd wordpress/ text naming
/home/lorna/downloads/wordpress/wp-includes/widgets.php:32 ❶
/home/lorna/downloads/wordpress/wp-includes/widgets.php:76 ❷
/home/lorna/downloads/wordpress/wp-includes/widgets.php:189 ❸
/home/lorna/downloads/wordpress/wp-includes/widgets.php:319 ❹
/home/lorna/downloads/wordpress/wp-includes/widgets.php:333I ❺
/home/lorna/downloads/wordpress/wp-includes/widgets.php:478 ❻
/home/lorna/downloads/wordpress/wp-includes/widgets.php:496 ❼
```

- ❶ *Avoid variables with short names like \$id.* (Staraj się nie stosować krótkich nazw zmiennych, takich jak \$id).
- ❷ *Classes shouldn't have a constructor method with the same name as the class.* (Konstruktor klasy nie powinien nazywać się tak samo jak zawierająca go klasa).
- ❸ *Avoid excessively long variable names like \$wp\_registered\_widgets.* (Staraj się nie nadawać zmiennym zbyt długich nazw, takich jak \$wp\_registered\_widgets).
- ❹ *Classes shouldn't have a constructor method with the same name as the class.* (Konstruktor klasy nie powinien nazywać się tak samo jak zawierająca go klasa).
- ❺ *Avoid excessively long variable names like \$wp\_registered\_widgets.* (Staraj się nie nadawać zmiennym zbyt długich nazw, takich jak \$wp\_registered\_widgets).
- ❻ *Avoid excessively long variable names like \$wp\_registered\_sidebars.* (Staraj się nie nadawać zmiennym zbyt długich nazw, takich jak \$wp\_registered\_sidebars).
- ❼ *Avoid extremely short variable names like \$n.* (Staraj się nie używać bardzo krótkich nazw zmiennych, takich jak \$n).

Narzędzie takie pewnie w każdym projekcie wyświetli jakieś błędy, ale warto z niego korzystać, aby móc zaobserwować ogólne tendencje. W punkcie 2. znajduje się komentarz, że nazwa konstruktora nie może być taka sama jak nazwa zawierającej go klasy, ale WordPress do niedawna był zgodny z PHP 4, więc nie jest to w nim błędem. Dostępne są jeszcze inne reguły, takie jak: metryki dotyczące rozmiaru kodu, elementy projektowe (np. wyszukiwanie użyci funkcji `eval()`) i znajdowanie nieużywanego kodu.

Wszystkie te narzędzia pozwalają lepiej zrozumieć zakres i kształt bazy kodu oraz ukazują obszary, które mogą wymagać poprawienia. W następnym podrozdziale pokażemy, jak sprawdzić, czy kod jest napisany zgodnie ze standardami kodowania.

## Standardy kodowania

Standardy kodowania to w niektórych zespołach programistycznych temat gorących dyskusji. Skoro rozmieszczenie wcięć i spacji nie ma żadnego znaczenia dla sposobu wykonywania kodu, to po co w ogóle definiować jakieś zasady, a potem się ich trzymać? Chodzi o to, że przyzwyczajamy się do określonego sposobu formatowania kodu i jeśli napotkamy kod napisany w taki sposób, łatwiej jest nam go zrozumieć.

Czasami jednak rozmieszczenie wszystkiego zgodnie ze standardem bywa bardzo trudne. Możesz przeczytać wszystkie wytyczne dotyczące projektu, do którego właśnie przystąpiłeś, ale i tak, gdy tylko zaczniesz pisać, zapominasz, gdzie powinien znajdować się każdy rodzaj nawiasu. Problem ten rozwiązuje się na dwa sposoby. Po pierwsze, poprzez odpowiednią konfigurację edytora, aby poprawnie stosował zakończenia wierszy oraz właściwie wstawiał tabulatory i spacje. Po drugie, możesz użyć specjalnego narzędzia do sprawdzania kodu, o nazwie PHP\_CodeSniffer.

## Weryfikacja kodu pod kątem standardów kodowania przy użyciu narzędzia PHP\_CodeSniffer

Najpierw trzeba zainstalować narzędzie na serwerze. To, czy zrobisz to na maszynie roboczej, czy na serwerze produkcyjnym, zależy od tego, jaką ilością zasobów dysponujesz. PHP\_CodeSniffer jest dostępny w PEAR<sup>1</sup> (szczegółowe informacje na temat korzystania z PEAR znajdują się w dodatku A). W wielu dystrybucjach Linuksa program ten jest również dostępny w postaci pakietu.



### Użycie narzędzia PHP\_CodeSniffer do analizy kodu JavaScript i CSS

Jeśli w projekcie znajdują się pliki JavaScript albo CSS, to je również można sprawdzić pod kątem zgodności z odpowiednimi standardami kodowania.

Po zainstalowaniu narzędzia można zacząć pracę. Sposób analizy kodu za jego pomocą przedstawimy na przykładzie bardzo prostej poniższej klasy:

```
class Robot {
    protected $x = 0;
    protected $y = 0;

    public function getCatchPhrase() {
        return 'Oto ja, intelekt przewyższający...';
    }

    public function Dance() {
        $xmove = rand(-2, 2);
        $ymove = rand(-2, 2);
        if($xmove != 0) {
            $this->x += $xmove;
        }
        if($ymove != 0) {
```

<sup>1</sup> [http://pear.php.net/package/PHP\\_CodeSniffer/](http://pear.php.net/package/PHP_CodeSniffer/).

```

        $this->y += $ymove;
    }
    return true;
}
}

```

Kod ten wygląda całkiem normalnie, prawda? Zobaczmy, co na jego temat powie nam PHP\_CodeSniffer. W tym przykładzie użyjemy standardu PEAR:

```
phpcs --standard=PEAR robot.php
```

```
FILE: /home/lorna/data/personal/books/Sitepoint/PHPPro/qa/code/robot.php
```

```
-----
FOUND 10 ERROR(S) AND 0 WARNING(S) AFFECTING 6 LINE(S)
-----
```

```

 2 | ERROR | Missing file doc comment
 4 | ERROR | Opening brace of a class must be on the line after the definition
 4 | ERROR | You must use "/*" style comments for a class comment
 8 | ERROR | Missing function doc comment
 8 | ERROR | Opening brace should be on a new line
12 | ERROR | Public method name "Robot::Dance" is not in camel caps format
12 | ERROR | Missing function doc comment
12 | ERROR | Opening brace should be on a new line
15 | ERROR | Expected "if (...) {\n"; found "if(...) {\n"
18 | ERROR | Expected "if (...) {\n"; found "if(...) {\n"
-----

```

Popełniliśmy 10 błędów — biorąc pod uwagę, że kod składa się tylko z 10 wierszy, nie jest to dobrym wynikiem. Jeśli jednak przyjrzyj się dokładniej tym danym, zauważysz, że niektóre błędy się powtarzają. Komunikaty o błędach dotyczą braku komentarzy (Missing file doc comment), niewłaściwego umiejscowienia nawiasów (Opening brace should be on a new line i Opening brace of a class must be on the line after the definition) oraz braku spacji za instrukcjami if (Expected "if (...) {\n"; found "if(...) {\n"). Oto poprawiona wersja tej klasy:

```

/**
 * Robot
 *
 * PHP Version 5
 *
 * @category Example
 * @package Example
 * @author Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license PHP Version 3.0 {@link http://www.php.net/license/3_0.txt}
 * @link http://sitepoint.com
 */
class Robot
{
    protected $x = 0;
    protected $y = 0;

    public function getCatchPhrase()
    {
        return 'Oto ja, intelekt przewyższający...';
    }
}

```



```

public function dance()
{
    $xmove = rand(-2, 2);
    $ymove = rand(-2, 2);
    if ($xmove != 0) {
        $this->x += $xmove;
    }
    if ($ymove != 0) {
        $this->y += $ymove;
    }
    return true;
}
}

```

Po ponownym sprawdzeniu tego kodu zauważymy, że większość błędów zniknęła. Do rozwiązania pozostała jeszcze tylko kwestia bloków komentarzy dla pliku i dla dwóch funkcji. Ponieważ w dalszej części rozdziału będziemy zajmować się pisaniem dokumentacji w kodzie, na razie zostawimy to bez poprawek.

## Przeglądanie przypadków naruszenia reguł standardów kodowania

PHP\_CodeSniffer oferuje kilka ciekawych opcji widoku, które pozwalają wygodnie oglądać raporty i uzyskiwać ogólny obraz bazy kodu. Można je wyświetlić w taki sam sposób jak wcześniejszy szczegółowy raport albo wyeksportować do innego formatu. Aby wygenerować raport podsumowujący, należy zastosować następujące polecenie:

```
phpcs --standard=PEAR --report=summary *
```

```
-----
PHP CODE SNIFFER REPORT SUMMARY
-----
```

FILE		ERRORS	WARNINGS
...e/eventscontroller.php	93	10	
...e/rest/index.php	29	3	
...e/rest/request.php		4	0

```
-----
A TOTAL OF 126 ERROR(S) AND 13 WARNING(S) WERE FOUND IN 3 FILE(S)
-----
```

Te wygenerowane dla usługi RESTful z rozdziału 3. dane pozwalają zorientować się, jak działa opisywana funkcja. W raporcie tym podane są: liczba błędów i ostrzeżeń znalezionych w poszczególnych plikach oraz suma wszystkich znalezionych błędów i ostrzeżeń. Raport można zapisać w kilku formatach, m.in. w CSV.

Jednym z popularnych formatów jest ten używany przez narzędzie do formatowania kodu Javy, o nazwie Checkstyle<sup>2</sup>. PHP\_CodeSniffer może generować dane w takim samym formacie jak Checkstyle (czyli XML), dzięki czemu można je wyświetlić przy użyciu dowolnego narzędzia obsługującego ten format. Zazwyczaj funkcji tej używa się w połączeniu ze środowiskiem ciągłej

<sup>2</sup> <http://checkstyle.sourceforge.net/>.

integracji, które generuje te dane regularnie i prezentuje je w formie internetowej. Ponadto wyświetlany jest wykres przedstawiający liczby błędów i ostrzeżeń wraz z informacją o tym, które usterki poprawiono, a które są nowe.

## Standardy kodowania w narzędziu PHP\_CodeSniffer

Narzędzie PHP\_CodeSniffer ma standardowo wbudowaną obsługę kilku standardów kodowania i pozwala na utworzenie i doinstalowanie nowych. Aby sprawdzić, jakie standardy są dostępne, należy wykonać polecenie `phpcs -i`:

```
phpcs -i
The installed coding standards are MySource, PEAR, Squiz, PHPCS and Zend
```

Jednym z najpowszechniejszych jest standard PEAR, z którego korzysta większość zespołów programistycznych. Standardy Zend nie są aktualnie standardem Zend Framework (w Zend Framework używa się specjalnie dostosowanej wersji standardu PEAR). Squiz<sup>3</sup> to całkiem dobry standard, ale ma bardzo restrykcyjne zasady dotyczące stosowania pustych wierszy, przez co trudno go używać na co dzień.

Kluczem do efektywnego posługiwania się standardami kodowania jest wybranie jednego z nich i korzystanie z niego, a nie gadanie o nim, ponieważ najważniejsze jest to, aby w ogóle **trzymać się** jakiegoś standardu! Spór dotyczący tego, czy otwierająca klamra powinna znajdować się w tym samym wierszu co instrukcja, czy w następnym, jest tak jałowy jak dyskusja o tym, czy lepszy jest edytor Vim, czy Emacs. Tych kwestii nie da się ostatecznie rozstrzygnąć.

Może się jednak zdarzyć, że podczas pracy nad programem wyniknie konieczność dostosowania albo rozluźnienia używanego standardu. Na przykład w projektach typu open source można zrezygnować z oznaczania autora w komentarzach, ponieważ takiej informacji precyzyjnie podać się nie da. Utworzenie własnego standardu nie jest trudne, zwłaszcza gdy wykorzystana jest już istniejąca zasada do własnych celów. Standardy programu PHP\_CodeSniffer zawierają szereg tzw. **niuchaczy** (ang. *sniff*), z których każdy wykonuje jedno ściśle określone zadanie, np. sprawdza, czy między instrukcją `if` a nawiasem jej warunku znajduje się spacja. Istniejące niuchacze można bardzo łatwo zmodyfikować, aby utworzyć własny nowy standard kodowania.

## Dokumentacja i kod

---

Dla większości programistów pisanie dokumentacji to prawdziwa katorga. Jednym ze sposobów na ułatwienie sobie tej pracy jest pisanie dokumentacji bezpośrednio wewnątrz kodu, w formie komentarzy. Dzięki temu patrząc na kod, widzi się od razu jego opis.

Wszystkie funkcje i klasy powinny być opatrzone komentarzami. Gdy wprowadza się jakieś zmiany w kodzie, można na bieżąco odpowiednio zmodyfikować jego dokumentację. Narzędzia weryfikujące zgodność ze standardami kodowania informują, gdzie brakuje komentarzy, dzięki czemu łatwiej jest pamiętać o konieczności ich wstawienia.

<sup>3</sup> <http://www.squizlabs.com/php-codesniffer>.

Ponieważ składnia komentarzy jest ściśle określona (o czym przekonaliśmy się w części „Weryfikacja kodu pod kątem standardów kodowania przy użyciu narzędzia PHP\_CodeSniffer”), można je pobrać z pliku i zamienić w prawdziwą dokumentację. Oto przykładowa klasa zawierająca wszystkie niezbędne komentarze:

```
/**
 * klasa Robot
 *
 * PHP Version 5
 *
 * @category Example
 * @package Example
 * @author Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license PHP Version 3.0 {@link http://www.php.net/license/3_0.txt}
 * @link http://sitepoint.com
 */

/**
 * Robot
 *
 * PHP Version 5
 *
 * @category Example
 * @package Example
 * @author Lorna Mitchell <lorna@lornajane.net>
 * @copyright 2011 Sitepoint.com
 * @license PHP Version 3.0 {@link http://www.php.net/license/3_0.txt}
 * @link http://sitepoint.com
 */
class Robot
{
    protected $x = 0;
    protected $y = 0;

    /**
     * pobranie typowego komentarza tego znaku
     *
     * @return string komentarz
     */
    public function getCatchPhrase()
    {
        return 'Oto ja, intelekt przewyższający...!';
    }

    /**
     * Przesuwa znak o losową liczbę znaków.
     *
     * @return logiczna wartość true
     */
    public function dance()
    {
        $xmove = rand(-2, 2);
        $ymove = rand(-2, 2);
        if ($xmove != 0) {
            $this->x += $xmove;
        }
    }
}
```

```

    }
    if ($ymove != 0) {
        $this->y += $ymove;
    }
    return true;
}
}

```

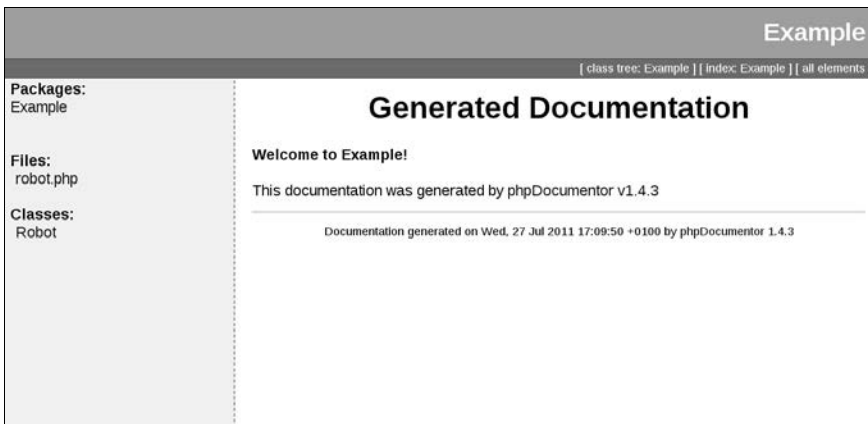
Większość środowisk programistycznych ma funkcję generowania szkieletu dokumentacji na podstawie deklaracji klas i metod, nazw parametrów itp. Później wystarczy tylko dodać brakujące informacje dotyczące przeznaczenia zmiennych, ich typów, postaci itd. Narzędzia wspomagające ten proces są bardzo pomocne, zatem nie masz żadnej wymówki, żeby się od ich używania wymigać!

## Narzędzie phpDocumentor

Narzędzi do zamiany komentarzy na dokumenty jest wiele. Jednym z nich, mającym ugruntowaną pozycję w środowisku, jest program phpDocumentor<sup>4</sup>, który można zainstalować poprzez bibliotekę PEAR (więcej na ten temat piszemy w dodatku A). Aby wygenerować dokumentację dla naszego bardzo prostego projektu, instalujemy wymienione narzędzie, a następnie wpisujemy poniższe polecenie:

```
phpdoc -t docs -o HTML:Smarty:PHP -d .
```

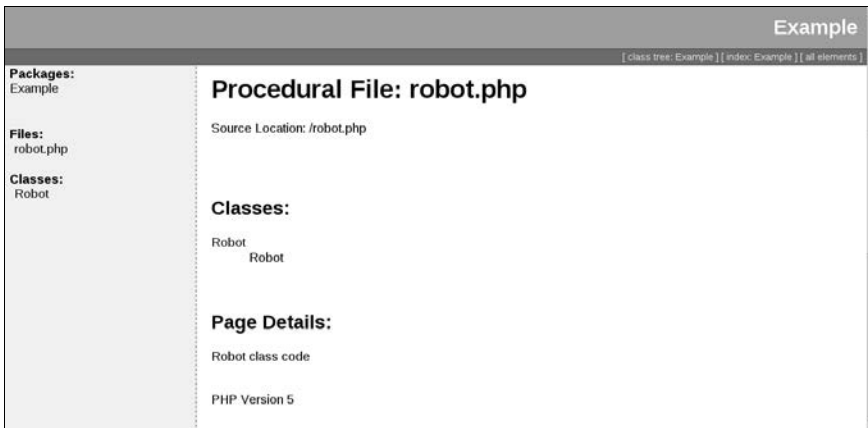
Pierwsza część polecenia to oczywiście nazwa programu. Za nią znajduje się kilka przełączników. Opcja `-t` określa katalog, w którym ma zostać zapisany wynik, `-o` wyznacza szablon, według którego ma zostać utworzona dokumentacja, a `-d` określa, gdzie znajduje się kod, dla którego ma zostać napisana dokumentacja — w tym przypadku jest to bieżący katalog. Po zakończeniu pracy programu można otworzyć stronę `docs/index.html` w przeglądarce internetowej (rysunek 8.1).



Rysunek 8.1. Dokumentacja wygenerowana przez program phpDocumentor

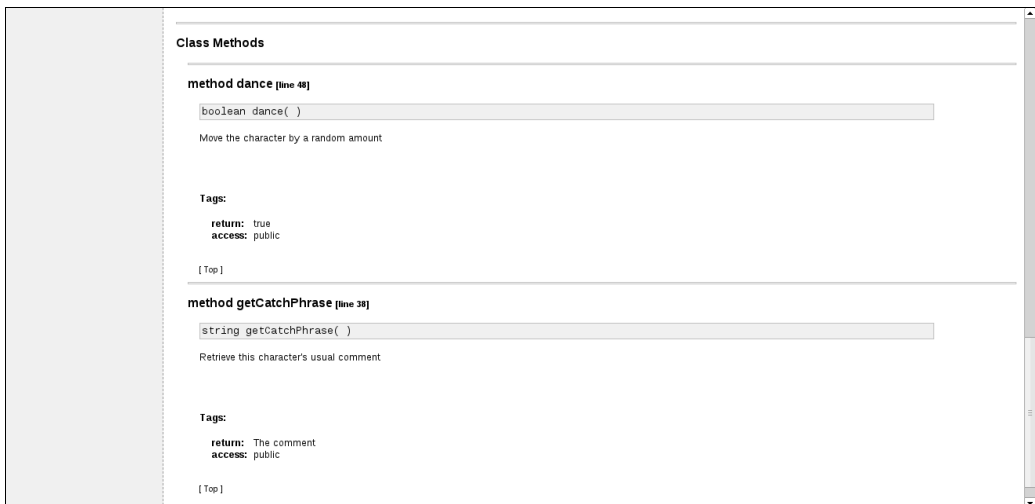
<sup>4</sup> <http://www.phpdoc.org/>.

Zapisane w tym pliku pobrane z kodu informacje można przeglądać na kilka sposobów, np. według zawartości plików, jak na rysunku 8.2.



Rysunek 8.2. Widok zawartości pliku w programie phpDocumentor

Informacje można także wyświetlać według klas, jak na rysunku 8.3.



Rysunek 8.3. Widok metod z klasy Robot

Przedstawione przykłady wydają się niezbyt bogate w treść, ale gdyby do narzędzia tego wprowadzić jakiś większy program, od razu dałoby się dostrzec wiele szczegółów. Co ważne, **nawet gdyby kod nie zawierał żadnych komentarzy**, phpDocumentor i tak wygenerowałby informacje o klasach, nazwach metod itp. Dzięki temu narzędzie to można wprowadzić do procesu budowy, aby mieć zawsze dostępną dokumentację API programu, nad którym się pracuje — w trakcie dalszych prac można dodać komentarze dokumentacyjne.

Program ten bardzo dobrze uzupełnia się z narzędziem PHP\_CodeSniffer, które ostrzega o brakujących komentarzach. Początkowo komentarzy jest dużo, ale możliwość sprawdzenia stanu rzeczy bardzo motywuje cały zespół do pracy.

## Inne narzędzia dokumentacyjne

Mimo że program phpDocumentor już od wielu lat jest standardowym narzędziem do tworzenia dokumentacji, nie uwzględniono w nim jeszcze nowości wprowadzonych w PHP 5.3. W celu zapewnienia tej luki pojawiły się nowe narzędzia tego typu, ale żadne z nich nie jest jeszcze wystarczająco dopracowane, aby można było je uznać za następcę starego programu. Ciekawie zapowiada się kilka projektów, np. DocBlox<sup>5</sup> i najnowsza wersja narzędzia Doxygen<sup>6</sup>, warto więc się trochę rozejrzeć, bo może uda Ci się znaleźć coś, co będzie odpowiadać Twoim wymaganiom.

## Kontrola źródła

---

Życzylibyśmy sobie, żeby w każdym projekcie był używany jakiś system kontroli kodu źródłowego, ale na wypadek gdybyś jeszcze niczego takiego nie używał albo był nowicjuszem w branży, w tym podrozdziale opisujemy wszystko od podstaw. Dowiesz się, dlaczego warto kontrolować kod źródłowy, jakie są dostępne narzędzia do robienia tego oraz jak utworzyć i skonfigurować repozytorium, aby odpowiadało Twoim potrzebom. Treść tej części rozdziału można ogólnie odnieść do wielu narzędzi tego typu, a przykłady tu prezentowane dotyczą systemów Subversion<sup>7</sup> i Git<sup>8</sup>. Panowanie nad kodem źródłowym i innymi zasobami projektu to klucz do sukcesu programisty. W tym podrozdziale znajdziesz wszystkie informacje potrzebne do osiągnięcia tego sukcesu.

Kontrola kodu źródłowego to nie tylko zapisywanie starszych wersji programu (choć to również się przydaje, gdy np. zauważysz, że zoczyłeś z kursu, albo klient stwierdzi, że poprzednia wersja programu bardziej mu się podobała). Dla każdej zmiany zapisywane są następujące informacje:

- kto dokonał zmiany,
- kiedy miało to miejsce,
- co dokładnie zmieniono,
- dlaczego to zrobiono<sup>9</sup>.

Z systemu kontroli kodu źródłowego warto korzystać nawet, gdy pracuje się nad projektem w pojedynkę, bez współpracy z innymi i bez tworzenia gałęzi. Repozytorium jest także centralnym magazynem kodu. Można w nim przechowywać pliki z kodem, przenosić te pliki na inne komputery, robić kopie zapasowe, używać repozytorium jako mechanizmu wdrażania (więcej na ten temat piszemy nieco dalej w tym rozdziale) i zawsze będzie wiadomo, że się pracuje na właściwej wersji kodu.

---

<sup>5</sup> <http://www.docblox-project.org/>.

<sup>6</sup> <http://www.stack.nl/~dimitri/doxygen/index.html>.

<sup>7</sup> <http://subversion.apache.org/>.

<sup>8</sup> <http://git-scm.com/>.

<sup>9</sup> Chyba że zezwolisz na wiadomości zatwierdzania typu „Poprawione”, które nie są zbyt pomocne.

System kontroli kodu to także ważne narzędzie ułatwiające współpracę. Umożliwia bezproblemowe wprowadzanie wielu zmian i uwalnia członków zespołu od konieczności wypytywania wszystkich w biurze, kto ostatnio wprowadził jakieś zmiany w programie, albo nazywania katalogów inicjałami programistów, aby dwie osoby równocześnie nie wprowadzały modyfikacji w tym samym kodzie!

## Praca z centralnym systemem kontroli wersji

W tekście pojawiło się kilka nowych słów, które mogą być niezrozumiałe, dlatego poniżej przedstawiamy definicje kilku pojęć.

<b>repozytorium</b> (ang. <i>repository</i> )	Miejsce przechowywania kodu
<b>zatwierdzenie zmian</b> (ang. <i>commit</i> )	Zarejestrowanie stanu zmian
<b>pobranie kodu</b> (ang. <i>check out</i> )	Pobranie kodu z repozytorium, aby na nim pracować
<b>kopia robocza</b> (ang. <i>working copy</i> )	Kod pobrany z repozytorium

Kod może być pobierany z repozytorium przez kilka osób jednocześnie. Każda z nich dokonuje w nim zmian, które następnie zatwierdza w repozytorium. Pozostałe osoby aktualizują kod, aby zachować wprowadzone zmiany w swoich kopiach roboczych. Relacje te przedstawiono na rysunku 8.4.



Rysunek 8.4. Kopie robocze pobrane z centralnego repozytorium

Czasami praca z systemem kontroli kodu może być trudna, zwłaszcza gdy członkowie zespołu nie mają na ten temat wystarczającej wiedzy. Wydaje się wówczas, że system, zamiast pomagać, tylko przeszkadza, a tak nie powinno być. Można jednak te kłopoty zminimalizować, postępując według prostych wskazówek. Oto kilka porad, które sformułowaliśmy na podstawie własnego doświadczenia:

- aktualizuj przed zatwierdzaniem;
- stosuj standardową konwencję nazywania projektów/gałęzi;
- często zatwierdzaj (przynajmniej raz dziennie) i często aktualizuj;
- przypominaj cały czas, kto nad czym pracuje (aby uniknąć dublowania pracy i konfliktów).

Wszystko, co zostało do tej pory napisane, to tylko teoria. W następnym podrozdziale przedstawimy praktyczny przykład na podstawie systemu Subversion. Natomiast Git i systemy rozproszone opisujemy nieco dalej.

## Kontrola źródła przy użyciu systemu Subversion

Większość organizacji wybiera do kontroli wersji oprogramowania system Subversion. Ostatnio można zaobserwować wzrost popularności systemów rozproszonych, ale wciąż jest miejsce dla prostych scentralizowanych narzędzi, zwłaszcza w zespołach, w których są młodzi programiści lub projektanci i większość osób pracuje w jednym miejscu lub w kilku miejscach. W każdym razie system Subversion i jego projekt mają się dobrze, a ich twórcy są gotowi na wszystko, aby dostarczyć produkt najwyższej jakości.

Przejrzymy polecenia, których najprawdopodobniej możesz potrzebować. Przede wszystkim musisz umieć pobrać kod (ang. *check out*), sprawdzić nowe zmiany oraz zatwierdzić własne zmiany (ang. *commit*):

```
$ svn checkout svn://repo/project
A   project/hello.php
Checked out revision 695.

$ svn update
A   project/readme
At revision 697.

$ vim hello.php
$ svn status
M   hello.php

$ svn commit -m "Fixed bug #42 by changing the wording"
Sending      hello.php
Transmitting file data .
Committed revision 698.
```

Najpierw pobraliśmy kod, aby mieć jego lokalną kopię roboczą. Jeśli chcesz ustawić jakieś opcje konfiguracyjne serwera, np. skonfigurować wirtualne hosty, powinieneś to zrobić właśnie teraz. Następne dwie czynności — aktualizacja i zatwierdzanie — są wykonywane wielokrotnie podczas pracy, a dodatkowo od czasu do czasu pobierane są zmiany od innych użytkowników. Po zakończeniu pracy należy wykonać ostateczną aktualizację, aby dokonać synchronizacji z lokalnym repozytorium, a następnie zatwierdzić zmiany. Pozostali członkowie zespołu zobaczą Twoje zmiany, gdy dokonają u siebie aktualizacji.

Tak wyglądają podstawowe zasady pracy. W ten sposób można zapanować nad kodem nawet w dużych zespołach programistycznych. Niestety, nie zawsze wszystko idzie tak dobrze! Jeśli dwie osoby dokonają zmian w tej samej części jakiegoś pliku, to Subversion nie będzie wiedział, która z tych zmian powinna być pierwsza, i poprosi o informację. W tym celu oznaczy plik jako **konflikt** (ang. *conflict*).

Przypuśćmy, że mamy plik o nazwie *hello.php* zawierający następujący prosty kod:

```
$greeting = "Witaj, świecie ";
echo $greeting;
```



Teraz zobaczmy, co się stanie, gdy dwie osoby dokonają zmian powodujących konflikt. Obaj programiści pobrali kod w wersji pokazanej powyżej. Następnie jeden z nich postanowił zmienić tekst powitania na mniej formalny:

```
$greeting = "Cześć, przyjacielu ";
echo $greeting;
```

Zmiana ta zostaje zatwierdzona w repozytorium w normalny sposób, ale w międzyczasie inny programista również wprowadził modyfikację, tak że kod wygląda teraz następująco:

```
$message = "Witaj, świecie ";
echo $message;
```

Próba zatwierdzenia zmian przez drugiego programistę nie powiedzie się, ponieważ jego pliki będą nieaktualne. Gdy obaj programiści dokonają aktualizacji, zostaną poinformowani o konflikcie, ponieważ zarówno w wersji przychodzącej, jak i w lokalnej kopii roboczej zmodyfikowano ten sam wiersz kodu.

Od Subversion 1.5 możliwe stało się interaktywne rozwiązywanie konfliktów, tzn. można edytować plik bezpośrednio podczas jego pobierania. Można także odłożyć zmiany na później i dokończyć aktualizację. W każdym razie w pliku zawierającym konflikty pojawi się następująca informacja:

```
<<<<<< .mine
$message = "Witaj, świecie ";
echo $message;
=====
$greeting = "Cześć, przyjacielu ";
echo $greeting;
>>>>>> .r699
```

Jeśli wykonasz w tym momencie polecenie `svn status`, zauważysz, że obok pliku *hello.php* znajduje się litera *C* oznaczająca, że wystąpił w nim konflikt. Ponadto pojawią się trzy nowe pliki: *hello.php.mine*, *hello.php.r698* oraz *hello.php.r699*. Zawierają one odpowiednio: Twój kod w wersji sprzed wykonania polecenia `svn update`, wersję z repozytorium po ostatniej aktualizacji lub ostatnim pobraniu oraz najnowszą wersję kodu z repozytorium.

Aby rozwiązać konflikt, należy otworzyć plik i ręcznie usunąć z niego informację o konflikcie, a następnie nanieść w nim odpowiednie poprawki. Po doprowadzeniu kodu do odpowiedniego stanu należy poinformować system o tym fakcie, wysyłając polecenie `resolved`:

```
svn resolved hello.php
```

Spowoduje to usunięcie znacznika oznaczającego konflikt, a także dodatkowo utworzonych plików. Dopóki konflikt nie zostanie rozwiązany, w pliku nie można zatwierdzać żadnych nowych zmian.



### Konflikty i zespoły

Konfliktów nie da się całkiem wyeliminować, zwłaszcza biorąc pod uwagę fakt, że Subversion nie może interpretować kodu PHP i nie wie, że to, co dla niego jest konfliktem na końcu pliku, to w rzeczywistości dwie nowe funkcje dodane przez dwie osoby. Jeśli zdarzenia takie mają miejsce bardzo często, może to być oznaką słabej komunikacji między pracownikami lub tego, że zbyt rzadko dokonują oni aktualizacji i zatwierżeń. Jeśli zauważysz, że w Twoim zespole konflikty występują regularnie, przyjrzyj się zasadom pracy i spróbuj poszukać rozwiązania, zmieniając jakieś procesy i zwyczaje.

## Projektowanie struktury repozytorium

W repozytorium Subversion można przechowywać wiele projektów, w których zazwyczaj tworzy się następujące katalogi: *branches*, *tags* oraz *trunk*<sup>10</sup>. W katalogu *trunk* przechowywana jest główna wersja kodu.

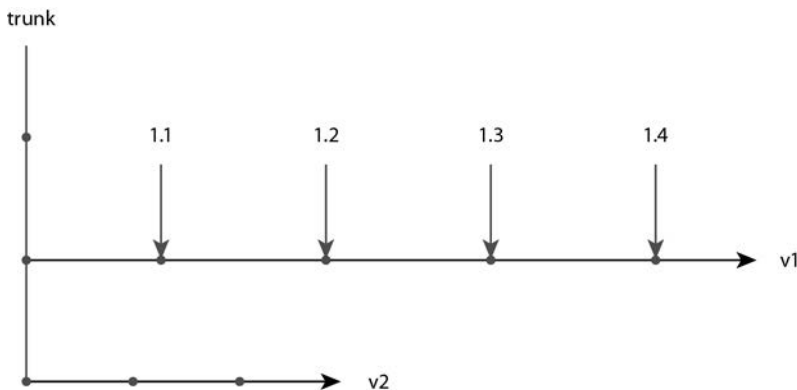
**Gałąź** (ang. *branch*) to kopia kodu. **Gałęzie** tworzy się po to, aby oddzielić pulę zmian od pnia głównego (ang. *trunk*), np. podczas pracy nad ważną funkcją. Bez rozgałęzienia programista pracujący nad tą funkcją nie mógłby współpracować z innymi programistami ani zatwierdzać zmian w repozytorium, dopóki nie miałby pewności, że jego funkcja jest gotowa i nie spowoduje uszkodzenia kodu kogoś innego. Dzięki utworzeniu gałęzi otrzymuje się bezpieczne środowisko pracy, w którym kod można zatwierdzać bez żadnych przeszkód, a także można współpracować z innymi tak, jak się chce.

**Znacznik** (ang. *tag*) to po prostu czytelna nazwa reprezentująca określony moment w czasie w repozytorium. Znaczników zazwyczaj używa się do oznaczania wybranych wersji, np. tej, którą się opublikowało.

Istnieje kilka szkół korzystania z katalogów *branches* i *tags*. W większości zespołów stosuje się jedno ze standardowych rozwiązań lub jego zmodyfikowaną wersję. Zobaczmy, na czym te rozwiązania polegają.

### Gałąź dla każdej wersji

Tego typu organizację katalogów stosuje się najczęściej w projektach opakowaniowych i w bibliotekach programistycznych. Jest katalog główny, ale wraz z pojawieniem się każdej wersji zostaje utworzona nowa gałąź. Za każdym razem, gdy wychodzi podwersja, dodaje się znacznik. Przedstawiono to schematycznie na rysunku 8.5.



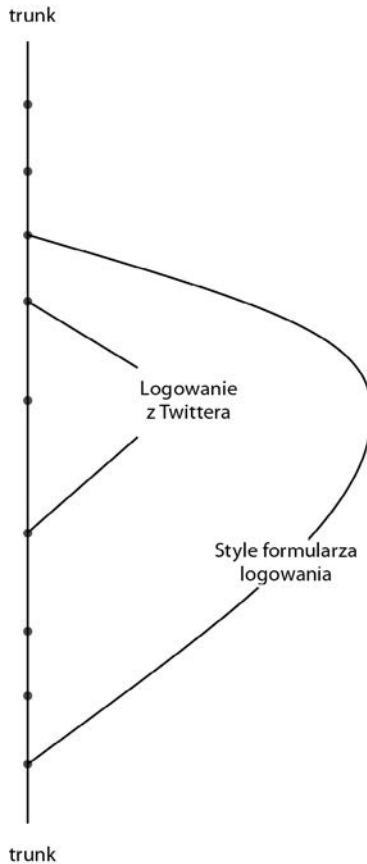
Rysunek 8.5. Struktura repozytorium z gałęziami i znacznikami w organizacji katalogów według wersji programu

<sup>10</sup> Jest to tylko zalecany standard, ale nie ma obowiązku tworzenia katalogów *branches* i *tags*.

W tym modelu nowe wersje są wyprowadzane w postaci gałęzi. Praca nad nową wersją odbywa się na pniu, po czym następuje wydanie głównej wersji, a mniejsze wersje i poprawki błędów są rozmieszczone wzdłuż gałęzi. Gdy w użyciu jest kilka wersji oprogramowania jednocześnie, poprawki usterek można też połączyć między gałęziami (więcej na temat łączenia piszemy nieco dalej).

## Gałąź dla każdej funkcji

Tę strategię zarządzania najczęściej stosuje się w projektach internetowych, głównie dlatego, że koszty przesyłania są bardzo niskie (zwłaszcza gdy korzysta się z automatycznych rozwiązań wdrażania, o których będzie mowa w podrozdziale „Automatyzacja procesu wdrażania”). W tym podejściu tworzy się nową gałąź dla każdej nowej funkcji programu. Większość zespołów godzi się na pewne szybkie poprawki bezpośrednio na pniu, ale tylko wtedy, kiedy dany zespół uważa, że jest to akceptowalne. Struktura repozytorium w tym wypadku wygląda tak jak na rysunku 8.6.



Rysunek 8.6. Repozytorium z gałęziami utworzonymi dla większych funkcji

Dla każdej nowej funkcji programu — np. dla opcji logowania do systemu przy użyciu Twittera — tworzona jest nowa gałąź. Później programiści zajmujący się tą funkcją mogą współpracować w normalny sposób, aż do jej ukończenia. Następnie można wszystko scalić z pniem.

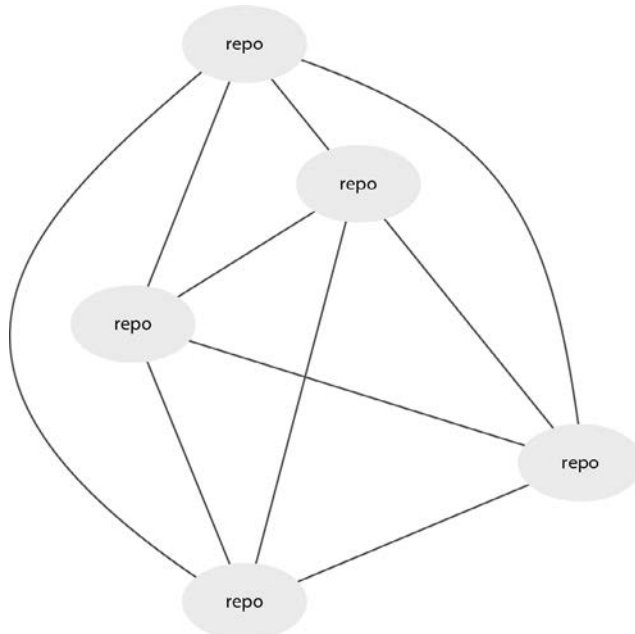
## Rozproszone systemy kontroli wersji

Coraz częściej zespoły programistyczne — głównie pracujące nad projektami otwartymi, ale zdarzają się i komercyjne — decydują się na używanie rozproszonych systemów kontroli wersji. Istnieje sporo programów tego typu, a najważniejsze z nich to:

- Git,
- Mercurial<sup>11</sup> (znany też pod nazwą Hg, która jest symbolem chemicznym pierwiastka rtęć — czyli *mercury* po angielsku),
- Bazaar<sup>12</sup> (znany też pod nazwą bzzr).

Wszystkie wymienione programy mają podobną funkcjonalność i każdy z nich działa na podobnych zasadach, dlatego też nie będziemy koncentrować się na żadnym konkretnym, lecz opiszemy koncepcje rozproszonej kontroli wersji w ujęciu ogólnym.

Cechą charakterystyczną odróżniającą systemy rozproszone od innych jest brak centralnego punktu. W systemie znajduje się wiele repozytoriów i każde z nich może wymieniać zatwierdzenia z pozostałymi. Na rysunku 8.4 przedstawiono schemat scentralizowanego repozytorium. W systemie rozproszonym nie pobiera się kodu z centralnego repozytorium, zamiast tego klonuje się je w celu utworzenia nowego własnego repozytorium. Zamiast kopii roboczych wszyscy mają repozytoria, a każde z nich jest połączone z wszystkimi pozostałymi. Schematycznie układ ten można przedstawić tak jak na rysunku 8.7.



Rysunek 8.7. Repozytoria typowego systemu rozproszonego

<sup>11</sup> <http://mercurial.selenic.com/>.

<sup>12</sup> <http://bazaar.canonical.com/en/>.

Użytkownicy mogą przysyłać zmiany ze swoich repozytoriów do innych oraz pobierać je z innych do swoich. Daje to szersze pole działania niż w scentralizowanych systemach, ale oznacza również, że jest więcej rzeczy, które trzeba wiedzieć, przez co nauka korzystania z systemów rozproszonych zazwyczaj trwa dłużej. Zwyczajowo jednemu z repozytoriów nadaje się rangę głównego, ale znajduje to odzwierciedlenie tylko w nazwie i nie pociąga za sobą nadania żadnych specjalnych właściwości. Po prostu głównym repozytorium jest to, które uwzględnia się w wykonywaniu kopii zapasowej i którego używa się jako bazy do wdrażania.

Jeśli planujesz zamienić system scentralizowany na rozproszony, musisz pamiętać o kilku istotnych różnicach między nimi. Po pierwsze, każde zatwierdzenie jest **serią zmian** (ang. *changeset*), a nie obrazem (ang. *snapshot*). Numer wersji oznacza zestaw zmian (coś w rodzaju łątki), a nie pełny eksport systemu. Druga ważna różnica dotyczy gałęzi. Ponieważ repozytorium jest przechowywane lokalnie, gałęzie możesz tworzyć w nim lokalnie albo oznaczyć je do udostępniania innym. Dlatego można tworzyć gałęzie na własny użytek, łączyć zmiany w udostępnianych gałęziach (albo je wyrzucać), a następnie przekazywać je do innych repozytoriów.

## Spółecznościowe narzędzia dla programistów

Nie można przy okazji omawiania narzędzia Git (i podobnych) nie wspomnieć o serwisach internetowych, które powstały w związku z nim, np. GitHub<sup>13</sup>. Są to usługi hostingu systemów kontroli kodu źródłowego, dzięki którym można śledzić aktywność wybranego programisty albo całych zespołów. Zazwyczaj w serwisach tych udostępniane są strony wiki i podsystemy typu issue tracker, czyli serwisy oferują właściwie wszystko, czego potrzeba do prowadzenia projektu programistycznego. Jednak najważniejszą cechą systemów rozproszonych, dzięki której zyskały popularność, jest możliwość sprawdzenia w dowolnej chwili, kto ma kopie repozytoriów i jakie zmiany ten ktoś wprowadza. Ponadto za pośrednictwem serwisów społecznościowych użytkownicy mogą wysyłać do nas **żądania pobrania** (ang. *pull request*) — prośby o wciągnięcie dokonanych przez nich zmian do naszej głównej gałęzi. Poza tym wiele z tych serwisów oferuje sieciowy interfejs do wykonywania tego typu scaleń.

Istnieją serwisy internetowe dla wszystkich rodzajów systemów kontroli kodu źródłowego, włącznie z Subversion. Są to doskonałe rozwiązania do pracy grupowej, a na dodatek większość z nich oferuje darmowe konta dla projektów open source i płatne do użytku komercyjnego.

## Kontrola kodu źródłowego przy użyciu narzędzia Git

Wcześniej poznaliśmy zasadę działania systemu Subversion. W tej części rozdziału dokonamy porównania tego narzędzia z rozproszonym systemem typu Git. Jedną z różnic między tymi typami rozwiązań jest stosowane nazewnictwo. W systemach rozproszonych repozytoria się **klonuje** (ang. *clone*), zamiast pobierać (ang. *check out*). Jeśli będziesz używać np. usługi GitHub, możesz na początku **rozwidlić** (ang. *fork*) repozytorium, aby utworzyć własną wersję, która będzie ogólnodostępna i w której będziesz mógł zapisywać swój kod — następnie klonujesz ją na swój komputer, aby móc na niej pracować.

<sup>13</sup> <http://github.com/>.

Do klonowania repozytoriów służy polecenie `clone`. Poniżej znajduje się przykładowe polecenie sklonowania z GitHub repozytorium otwartego projektu `joind.in`:

```
$ git clone git@github.com:lornajane/joind.in.git
Cloning into joind.in...
```

Program utworzy na Twoim komputerze nowy katalog o takiej samej nazwie jak repozytorium. Gdy do niego przejdiesz, znajdziesz w nim kompletny kod. Aby pobierać zmiany z innych repozytoriów, trzeba wiedzieć, czym są **źródła zdalne** (ang. *remotes*). Kontynuując poprzedni przykład: chcielibyśmy pobierać zmiany z głównego projektu `joind.in` z GitHub, z którego utworzyliśmy własne rozwidlenie repozytorium. W tym celu musimy go dodać jako źródło zdalne, a następnie pobrać z niego zmiany:

```
$ git remote add upstream git@github.com:joindin/joind.in.git
$ git remote
origin
upstream
```

Dodaliśmy główne repozytorium projektu `joind.in` jako źródło zdalne o nazwie `upstream` (jest to bardzo przydatny zwyczaj). Gdy wpiszemy polecenie `git remote` bez żadnych dodatkowych argumentów, otrzymamy listę wszystkich zdalnych źródeł znanych Git, włącznie z naszym źródłem `upstream` i źródłem `origin`, czyli tym, z którego je sklonowaliśmy. Aby pobrać zmiany z repozytorium `upstream`, należy użyć polecenia `pull`:

```
$ git pull origin master
```

Argumentami tego polecenia są nazwa źródła zdalnego i nazwa gałęzi, z której chcemy pobrać zmiany. Własnych zmian można dokonywać poprzez standardowe edytowanie plików. Aby jednak takie pliki zostały uwzględnione w zatwierdzeniu, w Git trzeba jeszcze je do niego dodać. Za pomocą polecenia `git status` można sprawdzić, co zostało zmienione, które pliki nie są śledzone oraz które zostały dodane do zatwierdzenia przy najbliższej okazji:

```
$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   index.php
#
no changes added to commit (use "git add" and/or "git commit -a")
```

```
$ git add index.php
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   index.php
#
```

```
$ git commit -m "added comments to index.php"
```

W tym przykładzie użyliśmy polecenia `git status`, aby zobaczyć, co zostało zmienione i dowiedzieć się, co dodaliśmy. Po zatwierdzeniu pliku można się przekonać, że zmiany zostały uwzględnione w wyniku polecenia `git log`, ale nadal są tylko w naszym lokalnym repozytorium. Aby przesłać je do repozytorium zdalnego (w tym przypadku do repozytorium GitHub), musimy je tam „wepchnąć” za pomocą polecenia `git push`. Domyślnie wysyła ono zmiany z lokalnego repozytorium do tego, którego jest klonem.

## Repozytorium jako centrum procesu budowy

Wiele z narzędzi, które zostały opisane w tym rozdziale, a także narzędzia testowe, najlepiej jest obsługiwać automatycznie. Niektóre na przykład powinny się uruchamiać za każdym razem po dokonaniu zatwierdzenia (np. testy i weryfikatory standardów kodowania). Potrzebny Ci też będzie jeden z systemów automatycznego wdrażania (systemom tym poświęcony jest następny podrozdział). Dzięki umieszczeniu kodu źródłowego w systemie kontroli źródła wszystkie te narzędzia wiedzą, skąd pobierać kod i jak prezentować zmiany w bieżącej wersji.

## Automatyzacja procesu wdrażania

Jak wysłać aplikację na platformę użytkową? Wiele osób odpowie, że trzeba użyć FTP albo SVN w celu pobrania na serwer nowych plików. Obie metody mają tę wadę, że powodują niewielkie zakłócenia podczas trwania procesu i nie umożliwiają cofnięcia operacji.



### Unikaj artefaktów kontroli źródła na platformach użytkowych

Podczas wysyłania plików z systemu kontroli źródła na platformę użytkową należy zachować szczególną ostrożność. Systemy te przechowują informacje o zmianach lokalnie, więc gdyby Twój serwer udostępniał je publicznie, mógłbyś przez przypadek udostępnić więcej informacji o swoim kodzie, niż byś chciał. Jeśli na przykład używasz systemu Subversion, to do wirtualnego hosta albo pliku `.htaccess` musisz dodać regułę zabraniającą serwowania wszelkich plików zawierających w ścieżce ciąg `.svn`.

## Natychmiastowe przełączanie na nową wersję

Lepszym rozwiązaniem kwestii wdrażania jest skonfigurowanie hosta tak, aby odwoływał się do **dowiązania symbolicznego** (`symlink`<sup>14</sup>) zamiast do zwykłego katalogu. Następnie wyślij kod na serwer i ustaw dowiązanie na jego katalog. Gdy będziesz gotów do wdrożenia nowej wersji, wyślij nowy kod na serwer i przygotuj go. Jeśli musisz skopiować albo dołączyć jakieś inne pliki konfiguracyjne lub zrobić cokolwiek innego, jest to odpowiedni moment na wykonanie tych czynności. Gdy wszystko będzie już gotowe, możesz po prostu przestawić dowiązanie symboliczne na nowy kod, nie powodując ani chwili przestoju.

Stosując tę metodę, masz również możliwość wycofania zmian. Jeśli stanie się coś nieprzewidywalnego i będzie trzeba wrócić do poprzedniej wersji aplikacji, wystarczy tylko przestawić dowiązanie symboliczne na stary katalog.

<sup>14</sup> <http://php.net/manual/en/function.symlink.php>.

## Zarządzanie zmianami w bazie danych

Jest to wyjątkowo trudna kwestia i mimo że bardzo byśmy chcieli przedstawić jakieś doskonałe rozwiązanie, tak naprawdę nie ma metody odpowiedniej dla każdego przypadku. Większość rozwiązań działa na podobnej zasadzie, tzn. polega na zapisywaniu ponumerowanych łatek do bazy danych, zapamiętywaniu, jaki numer w danej chwili nas interesuje, i zestawieniu tych dwóch wartości podczas aktualizacji wersji.

Wyjaśnimy to na przykładzie prostej bazy danych, której definicja znajduje się na poniższym listingu:

```
-- init.sql
CREATE TABLE categories
(id int PRIMARY KEY auto_increment,
name VARCHAR(255));

-- seed.sql
INSERT INTO categories (name) values ('Kids');
INSERT INTO categories (name) values ('Cars');
INSERT INTO categories (name) values ('Gardening');
```

Gdybyśmy chcieli zmienić schemat tej bazy, najpierw musielibyśmy opracować jakiś sposób zarządzania danymi tej aktualizacji. W tym przykładzie struktury łatki zostaną dodane jako sama łatka, dzięki czemu możesz to rozwiązanie wykorzystać we własnej bazie danych, jeśli chcesz zacząć formalne zarządzanie jej modyfikacjami. Najpierw tworzymy tabelę aktualizacyjną w pliku o nazwie *patch00.sql*:

```
CREATE TABLE patch_history (
patch_history_id int primary key auto_increment,
patch_number int, date_patched timestamp);

INSERT INTO patch_history SET patch_number = 0;
```

Teraz utworzymy pierwszą łatkę (w pliku *patch01.sql*), na podstawie której zilustrujemy sposób użycia tabeli *patch\_history*:

```
ALTER TABLE categories ADD COLUMN description varchar(255);

INSERT INTO patch_history SET patch_number = 1;
```

Utworzyliśmy tabelę o nazwie *patch\_history*, w której będzie można znaleźć informacje dotyczące tego, które łatki zostały zastosowane i kiedy to zrobiono. Są to bardziej precyzyjne dane niż tylko informacje o aktualnym poziomie łatki, które mogą być przydatne, gdy na przykład wprowadzenie któregoś z łatek nie powiodło się, a my dowiedzieliśmy się o tym dopiero po jakimś czasie. Umieszczając instrukcje dotyczące historii łatek na końcu plików z łatkami, mamy pewność, że będą one zastosowane tylko wtedy, kiedy pozostałe instrukcje zostaną wykonane pomyślnie.

W powyższym przykładzie wykonywana jest instrukcja `ALTER TABLE`. Dzięki umieszczeniu kodu SQL w plikach łatek i uruchomieniu tych ostatnich na roboczej bazie danych uzyskuje się zapis wszystkich dokonanych zmian. Jest to bardzo ważne, aby móc potem replikować te zmiany na innych platformach — zarówno roboczych, jak i produkcyjnych.



Jednym z najważniejszych aspektów zarządzania modyfikacjami bazy danych jest szansa **cofnięcia operacji** — powinno się mieć możliwość zarówno cofania zmian, jak i ich dokonywania. Trudność tę można w prosty sposób rozwiązać, pisząc dla każdej zmiany po dwie instrukcje SQL — jedną do wykonania zmian, drugą do ich cofania. Niestety, nie zawsze jest to możliwe, ponieważ np. instrukcje usuwania kolumn i ogólnie operacje powodujące usunięcie czegoś nie mogą być cofnięte.

Istnieje wiele narzędzi wspomagających zarządzanie zmianami baz danych. Można je znaleźć zarówno w niektórych frameworkach, jak i w narzędziach wdrożeniowych. Niezależnie jednak od tego, na co się zdecydujesz, pamiętaj, że system jest tak dobry jak dostarczane do niego informacje — jego działanie w pełni zależy od tego, czy zostaną mu przekazane pełne i poprawne zestawy łatek wraz z poprawnymi danymi dotyczącymi historii zmian.

## Automatyzacja wdrażania i plik konfiguracyjny Phing

Kilka razy wspomnieliśmy o automatyzacji procesu wdrażania. Teraz opiszemy dokładnie, jak to zrobić. Rozwiązanie tego typu trzeba zawsze dokładnie przemyśleć i zaplanować, ale gdy się już je skonfiguruje, pozwala ono zaoszczędzić wiele czasu i uniknąć niejednego błędu. Zastanów się nad następującymi kwestiami:

- Ile czasu zajmuje wdrażanie bazy kodu?
- Jak często popełniasz przy tym błędy?
- Jak często wdrażasz kod?
- Jak często byś to robił, gdyby to trwało krótko i nie było kłopotliwe?

Większość zespołów programistycznych zbyt nisko szacuje ilość czasu, jaki spędza na wdrażaniu (dla zabawy oszacuj, ile czasu zajmuje to Tobie, a następnie zmierz rzeczywisty czas podczas najbliższego wdrażania), i nie zdaje sobie sprawy ze szkód, jakie niosą błędy popełnione podczas wykonywania procesów, w których kilka czynności musi zostać wykonanych w ściśle określonym porządku. Zastosowanie wypróbowanego i przetestowanego procesu wdrażania aplikacji pozwala pozbyć się wielu potencjalnych problemów, a co więcej, jako że wdrażanie to już faza utrzymania programu, umożliwia zmniejszenie kosztów.

Najprostszy system automatycznego wdrażania składa się z szeregu skryptów, które wykonują podstawowe zadania. Typowy skrypt może działać według następującego schematu:

1. Oznakowanie i eksport kodu z systemu kontroli wersji.
2. Kompresja kodu do pliku TAR, przesłanie go na serwer i dekompresja.
3. Zastosowanie łatek do bazy danych, jeśli jest taka potrzeba.
4. Utworzenie łączy do elementów projektu znajdujących się poza katalogiem głównym, np. katalogów do odbierania plików wysyłanych przez użytkowników czy do plików konfiguracyjnych.
5. Przetawienie dowiązania symbolicznego na nową bazę kodu.
6. Opróżnienie buforów i ponowne uruchomienie serwerów.
7. Wizyta w barze i wypicie piwa.

Sposobów realizacji tego rozwiązania jest wiele: od własnoręcznie napisanych skryptów powłoki, po płatne oprogramowanie pisane na zamówienie. W ramach przykładu przedstawimy Phing<sup>15</sup>, narzędzie napisane w PHP przeznaczone do pracy z projektami w tym języku. Program ten ma wiele wtyczek usprawniających jego działanie oraz zawiera własne narzędzie do zarządzania bazą danych, o nazwie dbdeploy.

Plik konfiguracyjny Phing ma format XML i domyślnie nosi nazwę *build.xml*. Należy w nim wpisać nazwę projektu i zdefiniować serię zadań tego projektu. Można także zaznaczyć, które z nich mają być wykonywane domyślnie. Na poniższym listingu znajduje się zawartość przykładowego pliku konfiguracyjnego z dokumentacji Phing:

```
<?xml version="1.0" encoding="UTF-8"?>

<project name="FooBar" default="dist">

    <target name="prepare">
        <echo msg="Making directory ./build" />
        <mkdir dir="./build" />
    </target>

    <target name="build" depends="prepare">
        <echo msg="Copying files to build directory..." />

        <echo msg="Copying ./about.php to ./build directory..." />
        <copy file="./about.php" tofile="./build/about.php" />

        <echo msg="Copying ./contact.php to ./build directory..." />
        <copy file="./contact.php" tofile="./build/contact.php" />
    </target>

    <target name="dist" depends="build">
        <echo msg="Creating archive..." />

        <tar destfile="./build/build.tar.gz" compression="gzip">
            <fileset dir="./build">
                <include name="*" />
            </fileset>
        </tar>
        <echo msg="Files copied and compressed in build directory  OK!" />
    </target>
</project>
```

Kod ten jest bardzo przejrzysty, mimo że napisany w formacie XML. Najpierw tworzy się element `project` i ustawia w nim domyślny cel. Następnie definiuje się cele dla projektów: `prepare`, `build` oraz `dist`. Domyślny cel to `dist` i jeśli zależy on od jakichś innych celów, to najpierw zostaną wykonane właśnie one.

---

<sup>15</sup> <http://phing.info/>.



### Przechowywanie skryptów wdrożenia w kodzie

Dla każdego projektu musi być utworzony osobny plik *build.xml*, jeśli jednak tworzysz podobne serwisy, to zapewne w każdym z nich zastosujesz ten sam szkielet. Dobrym zwyczajem jest umieszczenie konfiguracji wdrażania w kodzie, ponieważ baza ta należy do projektu. Jest jego składnikiem, podobnie jak łatki do bazy danych, ale umieszczonym poza katalogiem głównym dokumentów.

Aby użyć narzędzia Phing, należy zastosować polecenie `phing`. Jeśli nie zostaną podane żadne argumenty, program wykona domyślny cel. Jeśli poda się konkretny cel, zostanie on wykonany zamiast domyślnego, np.:

```
phing prepare
```

Istnieje bardzo wiele gotowych zadań dla programu Phing, których, po uprzedniej konfiguracji, można użyć na swoim serwerze w celu dostosowania do własnych potrzeb. Narzędzie Phing potrafi uruchamiać testy jednostkowe, weryfikować standardy kodowania oraz używać większości innych narzędzi do analizy statycznej kodu. Można także za pomocą znacznika `exec` wykonać dowolne polecenie wiersza poleceń. Dzięki temu narzędzie to da się dostosować do potrzeb każdego projektu.

## Gotowi do wdrażania

W tym rozdziale opisaliśmy narzędzia do kontroli wersji i standardów kodowania oraz systemy automatycznego wdrażania aplikacji na serwerze. Poruszyliśmy też temat ciągłej integracji i serwera budowy. Każdy zespół programistyczny zapewne skorzysta z mieszanki przedstawionych tu rozwiązań, aby skonfigurować odpowiednie środowisko pracy, przystosowane do potrzeb projektu i do osób, które będą nad nim pracować.

Narzędzia, których opis znajduje się powyżej, pomogą Ci w większości prac, ale zastosowanie ich wszystkich naraz może być bardzo trudne. Dlatego najlepiej zrobisz, jeśli przejrzysz ten rozdział jeszcze raz i na początek wybierzesz tylko jedno z przedstawionych rozwiązań. Po pół roku, gdy nowy składnik już się „zaaklimatyzuje”, wybierz coś innego i powtórz czynności, które opisaliśmy.



# Skorowidz

---

\$\_COOKIE, 157  
\$\_GET, 109  
\$\_GET, 88, 109, 157  
\$\_POST, 88, 109, 157  
\$\_REQUEST, 157  
\$\_SERVER, 88  
\$\_SERVER['HTTP\_HOST'], 152  
\$\_SERVER['PHP\_SELF'], 155  
\$\_SESSION, 157  
\$captureScreenshotOnFailure, 232  
\$db\_conn, 54  
\$this, 21, 23  
.htaccess, 108, 257

## A

ab, 233, 235  
AcceptPathInfo, 154, 155  
adres IP, 169  
    falszowanie, 162  
Ajax, 100, 102  
aktualizacja, 250  
alpha, 276  
ALTER TABLE, 66, 258  
analiza statyczna kodu, 237  
Apache, 141, 154, 171, 175, 233  
ApacheBench, 173  
APC, 181, 186, 201, 203  
API, 75, 185, 247  
    Flickr, 97  
APT, 263  
architektura uslugowa, 76  
archiwum kodu, 16  
asercja, 207, 221, 230  
atak silowy, 167  
auto\_increment, 52  
autoloader, 208  
automatyczna inkrementacja, 52  
automatyczne ladowanie, 22  
AVG, 71  
AWS, 213

## B

Bazaar, 254  
BDD, 214, 217  
Behat, 214  
benchmark, 173, 179  
beta, 276  
bezpieczenstwo aplikacji, 152  
bezstanowy, 50  
brute force attack, 167  
bufor, 187  
    czyszczenie, 192  
buforowanie, 89, 185

## C

CAPTCHA, 169  
cechy, 119  
Charles Proxy, 96  
ciasteczko, 154, 170, 171, 176  
CMS, 49  
cofnięcie operacji, 259  
COUNT, 71, 126  
CPU, 195, 199  
cross-site Request Forgery, 156  
cross-site scripting, 153  
CRUD, 106, 126  
CSRF, 156, 159  
CSS, 171, 220, 228  
CSV, 201, 223, 243  
cURL, 83, 95

## D

DELETE, 59, 113  
deserializacja, 47  
destruktor, 21  
dev, 276  
Development Guide, 172  
diagnozowanie, 231  
DocBlock, 228  
    @scenario, 216

DocBlox, 248  
 dokumentacja  
   generowanie, 246  
   pisanie, 244  
 DOM, 79, 220  
 do-while, 126  
 dowiązanie symboliczne, 257, 259  
 Doxygen, 248  
 Drupal, 199, 203  
 DSL, 214  
 DSN, 53  
 dublery, 211  
 dziedziczenie, 27

## E

egzemplarz, 20  
 EXPLAIN, 65, 67  
 eXtensible Markup Language, 79  
 eZ Components, 268

## F

Facebook, 171, 173  
 Fail2ban, 169  
 FIFO, 287  
 filtrowanie, 152  
 final, 213  
 Firefox, 170, 233  
 Firesheep, 170, 171  
 Flickr, 97, 146  
 foreach, 126  
 FTP, 257  
 funkcja  
   \_\_autoload(), 282  
   anonimowa, 136  
   apc\_exists(), 186  
   apc\_fetch(), 186  
   apc\_store(), 186  
   array\_walk(), 44  
   assert(), 208  
   auto\_append\_file, 194  
   auto\_prepend\_file, 194, 198  
   count(), 37  
   date(), 81  
   error\_log(), 95  
   eval(), 240  
   GETAction(), 110  
   hash\_algos(), 166

  json\_decode(), 77, 109  
   json\_encode(), 77  
   md5(), 166  
   mysql\_escape\_string(), 57  
   mysql\_fetch\_array(), 143  
   ob\_flush(), 89  
   preg\_match(), 143  
   print\_r(), 98  
   rand(), 42  
   rdzenna, 221  
   session\_regenerate\_id(), 160  
   set\_error\_handler(), 43, 44  
   set\_exception\_handler(), 43  
   simplexml\_load\_file(), 81  
   simplexml\_load\_string(), 81  
   sprintf(), 150  
   urlSimilarator, 199  
   var\_dump(), 22, 47

## G

gałąź, 252  
   dla każdej funkcji, 253  
   dla każdej wersji, 252  
 GET, 82, 91, 175, 177, 200  
 Git, 248, 249, 254, 255, 256  
 GitHub, 197, 255, 256  
 Gluster, 186  
 Google Groups, 291  
 Google+, 173  
 graficzny interfejs użytkownika, 175  
 GROUP BY, 72  
 GUI, 201

## H

hartowanie kodu, 76  
 hermetyzacja, 19  
 HMAC, 167  
 Horde, 268  
 HTML, 17, 77, 102  
   struktura, 154  
 htmlentities(), 155  
 HTTP, 82, 233, 235  
   kod statusu, 87  
   żądanie, 177  
 HTTPS, 171

**I**

identyfikator sesji, 159  
 iframe, 104  
 indeks, 65  
 index.php, 108  
 INNER JOIN, 69  
 INSERT, 58, 59  
 instrukcja preparowana, 55, 164  
 interfejs, 37
 

- ArrayAccess, 281
- Countable, 37, 285
- Trackable, 38, 39

 IRC, 290, 291  
 iterator, 281, 282
 

- FilterIterator, 130
- LimitIterator, 131, 133
- łączenie, 132
- OuterIterator, 128, 132
- RecursiveIterator, 129
- RecursiveIteratorIterator, 133
- RegexIterator, 131

**J**

Java, 180, 221, 243
 

- serwer, 226

 JavaScript, 77, 114, 171, 228
 

- Object Notation, 77

 język
 

- kompilowany, 180
- skryptowy, 179
- specjalistyczny, 214

 JMeter, 173, 175  
 jQuery, 102, 228, 229  
 JSON, 77, 99, 100, 102, 114, 187

**K**

kanał, 279  
 katalog trunk, 252  
 KCacheGrind, 192  
 KISS, 114  
 klasa, 20
 

- Courier, 20, 25, 27
- DirectoryIterator, 283
- Exception, 40
- FileSystemIterator, 283
- My\_Calculator, 209

Parcel, 29  
 PDOStatement, 54, 56, 165  
 PigeonPost, 30  
 RecursiveDirectoryIterator, 283  
 RecursiveIteratorIterator, 283  
 Registry, 121  
 SoapClient, 93  
 SplFileInfo, 283  
 SplFileObject, 284  
 SplHeap, 287  
 SplQueue, 287  
 SplStack, 287  
 SplTempFileObject, 284  
 klauzula finally, 40  
 klucz
 

- główny, 51
- obcy, 67

 kod operacyjny buforowanie, 180  
 kolejka, 287  
 komentarze, 244
 

- generowanie, 246

 kompilacja, 264  
 kompilator, 180  
 konflikt, 250  
 konstrukcja try-catch, 40  
 konstruktor, 21  
 kontroler, 140  
 kopia robocza, 249

**L**

LEFT JOIN, 70  
 LIFO, 287  
 LIMIT, 126, 131  
 localhost, 53  
 lokalizatory, 228  
 losowa wartość, 158

**Ł**

łańcuchy wywołań, 32  
 łątką, 258

**M**

Mac OS X, 193  
 maszyna wirtualna, 180  
 MAX, 71  
 MD5, 166

Memcached, 184, 186  
 Mercurial, 254  
 metadane, 187  
 metapakiety, 280  
 metoda, 20
 

- \_\_autoload(), 22
- \_\_call(), 45, 228
- \_\_callStatic(), 46
- \_\_clone(), 32
- \_\_construct(), 20
- \_\_destruct(), 21
- \_\_get(), 36
- \_\_getFunctions(), 95
- \_\_set(), 36
- \_\_toString(), 46

 add(), 213  
 assertEquals(), 207  
 assertNotTitle(), 229  
 assertTiel(), 229  
 bindParam(), 58  
 bindValue(), 58  
 calculateShipping(), 34  
 DBConnection::getInstance(), 124  
 equalTo(), 212  
 errorInfo(), 62  
 execute(), 56, 58, 165  
 fetch(), 62  
 fetchAll(), 54  
 GET, 157  
 getConnection(), 222  
 getDataSet(), 223, 225  
 getLog(), 125  
 getShippingRateForCountry(), 34  
 getTitle(), 229  
 getTrackInfo(), 38  
 magiczna, 21  
 method(), 212  
 onSetUp(), 231  
 onTearDown(), 231  
 open(), 227  
 PDO::beginTransaction(), 63  
 PDO::commit(), 63  
 PDO::prepare(), 56, 60  
 PDO::query(), 54  
 PDO::rollback(), 63  
 PDOStatement::bindParam(), 58  
 PDOStatement::bindValue(), 57  
 PDOStatement::errorInfo(), 62  
 PDOStatement::fetch(), 54

POST, 158  
 prepare(), 56, 61, 165  
 render(), 149  
 reset(), 127  
 rewind(), 127  
 rowCount(), 59  
 runGiven(), 215  
 runThen(), 215  
 runWhen(), 215  
 setBrowser(), 227  
 setBrowserUrl(), 227  
 setHost(), 226  
 setPort(), 226  
 setSetUpOperation(), 231  
 setTearDownOperation(), 231  
 setTimeout(), 227  
 setUp(), 212, 219, 225, 231  
 ship(), 20  
 sizeof(), 285  
 sprawdzająca, 35  
 statyczna, 23  
 tearDown(), 225, 231  
 testActionGet(), 219  
 testDoStuff(), 225  
 ustawiająca, 35  
 waitForPageToLoad(), 230  
 will(), 212  
 metodyka czarnej skrzynki, 218  
 MIN, 71  
 Mockery, 213  
 mod\_rewrite, 141, 171  
 model, 148  
 modyfikator
 

- dostępu, 33
- private, 33, 34
- protected, 34
- public, 33

 MVC, 139, 217  
 MySQL, 49, 51, 70, 124, 186
 

- klient, 51

 MySQLi, 197

## N

nagłówek
 

- Accept, 90
- Content-Type, 90
- Expires, 187
- HTTP Location, 159



- Last-Modified, 187
- Timestamp, 199
- nagłówki żądań, 162
- narzędzie
  - ab, 233
  - Checkstyle, 243
  - dbdeploy, 260
  - Phing, 260
  - PHP\_CodeSniffer, 241, 244, 248
  - phpcpd, 239
  - phpDocumentor, 246, 247
  - phploc, 238
  - phpmd, 240
  - phpunit, 209
  - xhprof.profile, 194
- negocjacja treści, 90
- Netflix, 212
- new, 21
- NFS, 186
- niuchacz pakietów, 170
- normalizacja danych, 72
- NoSQL, 50, 184

## O

- obiekt, 20
  - dopasowujący, 212
  - PDOStatement, 61
- OFFSET, 126
- onclick, 102, 133
- onload, 133
- onmouseover, 133
- OOP, 19
- opcja
  - apc.stat, 183
  - servername, 198
- open source, 239, 255
- operator
  - instanceOf, 39
  - obiektowy, 23
  - przestrzeni nazw, 25
  - zakresu, 23

## P

- packet sniffer, 170
- parsowanie, 180
- partycja, 186
- PCRE, 143, 153

- PDO, 40, 49, 53, 60, 65, 165, 222
- PEAR, 206, 208, 241, 244, 246, 263, 276, 278, 280
  - instalowanie rozszerzeń, 268
  - kanał, 265, 266, 277
  - menedżer pakietów, 263
  - tworzenie pakietów, 272
- PECL, 181, 193, 263
  - ręczne instalowanie rozszerzeń, 269
- Phake, 213
- Phing, 259, 260, 261
  - konfiguracja, 260
- PHP
  - Architect, 289
  - Planet, 289
  - strumienie, 86
- php.ini, 183, 184
- PHP\_CodeSniffer, 244
- PHPDeveloper, 289
- phploc, 238
- PHPMachinist, 221
- phpMyAdmin, 51
- phpQuery, 220
- PHPSpec, 214
- PHPT, 205
- PHPUnit, 205, 206, 212, 214, 220, 221, 222, 224, 230, 268
- pień główny, 252
- Pirum, 277
- plik make, 271
- płatności online, 172
- płynny interfejs, 32
- plytkie kopiowanie obiektów, 32
- PNG, 232
- pobranie kodu, 249
- polecenie
  - channel-info, 266
  - clone, 256
  - config -set, 265
  - config -show, 265
  - configure, 270
  - COUNT(), 72
  - git log, 257
  - git push, 257
  - git remote, 256
  - git status, 256
  - mysqldump, 224
  - pear package, 275
  - phing, 261
  - phpcs, 244

- polecenie
  - phpize, 193
  - phpunit, 208
  - pull, 256
  - resolved, 251
  - svn status, 251
  - svn update, 251
- Polimorfizm, 29
- POST, 91, 97, 98, 109, 111, 168, 177, 200, 201
- późne ładowanie, 118
- private, 33, 213
- procedury składowane, 64
- profilowanie, 192
- programowanie
  - obiektywne, 13, 19
  - oparte na testach, 213
  - oparte na zachowaniach, 214
- protected, 33
- przechowywanie
  - danych, 50
  - haseł, 165
- przekazywanie przez referencję, 31
- przestrzeń nazw, 24
- public, 23, 33
- PUT, 109, 112
- Pyrus, 280

## Q

- QCachegrind, 192

## R

- RAID, 185
- rainbow tables, 166
- RAM, 185
- raport, 243
  - pokrycia kodu, 210
- reCAPTCHA, 169
- referencja, 31
- regexp, 229
- regexp, 229
- rekurencja, 128
- repozytorium, 248, 249
  - główne, 255
  - klonowanie, 255
  - rozproszone, 254
  - rozwidlenie, 255
  - scentralizowane, 249
- require, 22

- REST, 92, 106
- RESTful, 106, 107, 111, 243
- RewriteCond, 141
- RewriteRule, 141
- RIGHT JOIN, 70
- router, 139
- rozszerzenie
  - Database, 221, 230
  - DbUnit, 221
  - JUnit, 221
  - memcache, 184
  - pecl\_http, 85, 108
  - Xdebug, 192, 210, 271
- RPC, 92

## S

- Samba, 186
- SAN, 186
- SCSI, 185
- Secure Socket Layer, 171
- SELECT, 58, 67
- Selenium, 220, 226, 227, 228, 229, 231, 233
  - IDE, 233
- Selenium RC, 226
- Selenium Server, 226
- serializacja, 46, 47
- serwer, 233
- sesja, 159, 176
  - identyfikator, 160
  - przechwycenie, 159
  - uprowadzenie, 161
- session
  - fixation, 159
  - hijacking, 161
- session.cookie\_httponly, 162
- Set-Cookie, 83
- SHA-1, 166, 186
- SHA-256, 166
- Siege, 234, 235
  - plik konfiguracyjny, 235
- SimpleTest, 205
- SimpleXML, 79
- SimpleXMLElement, 98
- skalowanie, 204
- skrypt pośredniczący, 104
- słowo kluczowe
  - clone, 32
  - extends, 28

- public, 33
- throw, 41
- trait, 120
- use, 120
- SOA, 76
- SOAP, 82, 92, 114
- sól, 166
- SPL, 126, 281, 282
- spoofing, 162
- SQL, 185
  - injection, 163, 165
  - zapytanie, 163, 204
- SQLite, 124
- Squiz, 244
- SSL, 170
  - certyfiakat, 171
- stable, 276
- Stack Overflow, 292
- Standard PHP Library, 126
- static, 23, 213
- sterta, 287
- stopniowe ulepszanie, 102
- stos, 287
- struktura
  - klas, 26
  - SplDoublyLinkedList, 286
  - SplPriorityQueue, 288
- Subversion, 248, 249, 250, 255, 257
- SUM, 71
- SVN, 257
- symbol zastępczy, 56
- Symphony, 268
- system kontroli kodu, 249

## T

- tabela aktualizacyjna, 258
- tablica, 286
- TAR, 259, 272
- testowanie
  - jednostkowe, 205
  - obciążeniowe, 173, 233
  - systemowe, 226
- testy
  - funkcjonalne, 218
  - jednostkowe, 218
- tęczowe tablice, 166
- token CSRF, 158
- transakcja, 63

- try-catch, 40
- Twitter, 253, 290
- tworzenie
  - makiety, 212
  - zaślepki, 212

## U

- UML, 27
- Uncaught Exception, 43
- UPDATE, 58, 59
- usługa sieciowa, 75, 77

## W

- W3C, 229
- warstwa abstrakcji, 53
- wąskie gardło, 192
- wątki, 174
- wdrażanie, 259
- Web Developer, 170
- WebCacheGrind*, 193
- wersja kodu, 252
- weryfikacja, 152
- Wget, 270
- WHERE, 67, 164
- widok, 149
- wiki, 255
- WinCache, 183
- Windows, 193
- Wireshark, 96
- wirtualny host, 250, 278
- własność, 20
  - statyczna, 23
- WordPress, 238, 239, 240
- WSDL, 94
- wskazywanie typów parametrów, 29
- wydajność, 286
- wyjątek, 40, 41
  - PDOException, 54, 60
- wyrażenia regularne, 153
- wzorzec
  - fabryka, 124
  - iterator, 125
  - model-widok-kontroler, 139
  - obserwator, 133
  - rejestr, 120
  - singleton, 118
  - wstrzykiwanie zależności, 136

## X

Xampp, 53  
XHGui, 192, 197, 199, 201  
XHProf, 192, 193, 204, 264  
XML, 17, 77, 79, 98, 223, 239, 243, 260  
    Flat, 223  
    MySQL, 223  
    YAML, 223  
XML\_Parser, 265  
XML-RPC, 114  
XPath, 220, 228, 229  
XSS, 153, 161, 163

## Y

YUM, 263

## Z

zamknięcie, 136  
zasada tej samej domeny, 100  
zatwierdzanie, 250  
    zmian, 249

Zend, 244  
Zend Engine, 180  
Zend Framework, 24, 217, 220, 244, 268  
Zend\_Dom\_Query, 220  
złączenia  
    wewnętrzne, 69  
    zewnętrzne, 70  
zmiennie superglobalne, 88  
znacznik, 252  
zrzut ekranu, 231

## Ż

źródła zdalne, 256

## Ż

żądania, 174  
    na sekundę, 234  
żądanie pobrania, 255

# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ 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>

GRUPA WYDAWNICZA

 **Helion SA**

*Mistrz PHP. Pisz nowoczesny kod* to książka przeznaczona dla programistów PHP, którzy znają już podstawy tego języka i chcą rozwinąć swoje umiejętności, by tworzyć bardziej zaawansowane rozwiązania. Znajdziesz w niej cenne rady, jak udoskonalić swoje aplikacje serwerowe, oraz wszystko, czego potrzeba do stosowania najefektywniejszych technik obiektowych, zabezpieczania kodu czy pisania programów idealnie spełniających swoje zadania. W każdym rozdziale poznasz nowe sposoby wykonywania pewnych zadań oraz teorie leżące u podłoża stosowanych przez Ciebie technik.

Dzięki lekturze tej publikacji przemienisz się ze sprawnego programisty w pewnego siebie inżyniera — stosującego najlepsze praktyki programistyczne, pracującego szybko i solidnie. Autorzy przedstawiają praktyczne problemy i użyteczne rozwiązania, które zaprowadzą Cię na szczyt kariery! Jeśli szukasz możliwości scementowania całej swojej wiedzy i chcesz zdobyć solidne podstawy, ta książka jest dla Ciebie.

## PRAKTYCZNE PROBLEMY I UŻYTECZNE ROZWIĄZANIA, KTÓRE ZAPROWADZĄ CIĘ NA SZCZYT KARIERY!

Z tej książki dowiesz się, jak:

- tworzyć profesjonalne dynamiczne aplikacje na podstawie obiektowych wzorców programowania
- używać zaawansowanych narzędzi do oceny wydajności programów, aby zmaksymalizować ich możliwości
- stosować nowoczesne techniki testowania, pozwalające uzyskać niezawodny kod
- zabezpieczać programy przed atakami zewnętrznymi przy użyciu najskuteczniejszych technik
- używać funkcji dostępnych w bibliotekach i interfejsach programistycznych języka PHP

...i wiele więcej!

Nr katalogowy: 8 5 4 3

Księgarnia internetowa:  
<http://helion.pl>

Zamówienia telefoniczne:  
0 801 339900  
0 601 339900

**helion.pl**  
księgarnia internetowa

Sprawdź najnowsze promocje:  
• <http://helion.pl/promocje>  
Książki najchętniej czytane:  
• <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helion.pl/nawosci>

 **Helion**

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

 **sitepoint**

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

Cena 49,00 zł

ISBN 978-83-246-4472-8



Informatyka w najlepszym wydaniu