

# Czysta architektura

Struktura i design oprogramowania  
Przewodnik dla profesjonalistów

Robert C. Martin



Tytuł oryginału: Clean Architecture: A Craftsman's Guide  
to Software Structure and Design (Robert C. Martin Series)

Tłumaczenie: Wojciech Moch

ISBN: 978-83-283-4225-5

Authorized translation from the English language edition, entitled: CLEAN ARCHITECTURE: A CRAFTSMAN'S GUIDE TO SOFTWARE STRUCTURE AND DESIGN; ISBN 0134494164; by Robert C. Martin; published by Pearson Education, Inc, publishing as Prentice Hall. Copyright ©2018 by Pearson Education, Inc.

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 Pearson Education, Inc.

Polish language edition published by HELION S.A. Copyright © 2018.

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/czarch>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

---

# Spis treści

---

<b>Przedmowa</b> .....	<b>13</b>
<b>Wstęp</b> .....	<b>17</b>
<b>Podziękowania</b> .....	<b>21</b>
<b>O autorze</b> .....	<b>23</b>
<b>Część I. Wprowadzenie</b> .....	<b>25</b>
<b>Rozdział 1. Czym są projekt i architektura?</b> .....	<b>27</b>
Cel? .....	28
Studium przypadku .....	29
Wnioski .....	35
<b>Rozdział 2. Opowieść o dwóch wartościach</b> .....	<b>37</b>
Zachowanie .....	38
Architektura .....	38
Ważniejsza wartość .....	39
Macierz Eisenhowera .....	40
Walka o architekturę .....	41

**Część II. Zaczynij od podstaw.  
Paradygmaty oprogramowania ..... 43**

**Rozdział 3. Przegląd paradygmatów ..... 45**

Programowanie strukturalne .....	46
Programowanie obiektowe .....	46
Programowanie funkcyjne .....	46
Coś do przemyślenia .....	47
Wnioski .....	47

**Rozdział 4. Programowanie strukturalne ..... 49**

Dowód .....	50
Ogłoszenie szkodliwości .....	52
Dekompozycja funkcyjna .....	52
Brak formalnych dowodów .....	53
Metoda naukowa .....	53
Testy .....	54
Wnioski .....	55

**Rozdział 5. Programowanie obiektowe ..... 57**

Hermetyzacja? .....	58
Dziedziczenie? .....	61
Polimorfizm? .....	63
Wnioski .....	68

**Rozdział 6. Programowanie funkcyjne ..... 69**

Kwadraty liczb całkowitych .....	70
Niezmiennosc i architektura .....	71
Podzial zmienności .....	72
Strumień zdarzeń .....	74
Wnioski .....	75

**Część III. Reguły projektowe ..... 77**

**Rozdział 7. SRP — reguła jednej odpowiedzialności ..... 81**

Symptom 1. Przypadkowa duplikacja .....	83
Symptom 2. Złączenia .....	84
Rozwiązania .....	85
Wnioski .....	87

---

<b>Rozdział 8. Reguła otwarte-zamknięte .....</b>	<b>89</b>
Eksperyment myślowy .....	90
Kontrola kierunku .....	94
Ukrywanie informacji .....	94
Wnioski .....	95
<b>Rozdział 9. Zasada podstawień Barbary Liskov .....</b>	<b>97</b>
Jak używać dziedziczenia? .....	98
Problem z kwadratem i prostokątem .....	98
Zasada LSP i architektura .....	99
Przykład naruszenia zasady LSP .....	100
Wnioski .....	101
<b>Rozdział 10. Zasada rozdzielania interfejsów .....</b>	<b>103</b>
Zasada ISP i język .....	105
Zasada ISP i architektura .....	105
Wnioski .....	106
<b>Rozdział 11. Zasada odwrócenia zależności .....</b>	<b>107</b>
Stabilne abstrakcje .....	108
Fabryki .....	109
Komponenty konkretne .....	110
Wnioski .....	111
<b>Część IV. Zasady komponentów .....</b>	<b>113</b>
<b>Rozdział 12. Komponenty .....</b>	<b>115</b>
Krótka historia komponentów .....	116
Relokacje .....	119
Konsolidatory .....	119
Wnioski .....	121
<b>Rozdział 13. Spójność komponentów .....</b>	<b>123</b>
Zasada Reuse (Release Equivalence Principle) .....	124
Zasada Common Closure Principle .....	125
Zasada Common Reuse Principle .....	126
Diagram napięć dla zasad spójności komponentów .....	127
Wnioski .....	128

<b>Rozdział 14. Łączenie komponentów .....</b>	<b>131</b>
Zasada zależności niecyklicznych .....	132
Projekt typu top-down .....	138
Zasada stabilnych zależności .....	139
Zasada stabilnych abstrakcji .....	144
Wnioski .....	150
<b>Część V. Architektura .....</b>	<b>151</b>
<b>Rozdział 15. Czym jest architektura? .....</b>	<b>153</b>
Rozwój systemu .....	155
Wdrożenia .....	155
Działanie .....	156
Konservacja .....	157
Zachowywanie dostępnych opcji .....	157
Niezależność od urządzenia .....	159
Spam .....	160
Adresowanie fizyczne .....	162
Wnioski .....	163
<b>Rozdział 16. Niezależność .....</b>	<b>165</b>
Przypadki użycia .....	166
Działanie .....	166
Rozwój .....	167
Wdrożenia .....	167
Otwarte opcje .....	168
Oddzielanie warstw .....	168
Rozdzielanie przypadków użycia .....	169
Tryby rozdzielania .....	170
Możliwość niezależnego rozwijania .....	171
Niezależne wdrożenia .....	171
Duplikacja .....	171
Tryby rozdzielania (ponownie) .....	172
Wnioski .....	174
<b>Rozdział 17. Granice. Wyznaczanie linii .....</b>	<b>175</b>
Dwie smutne historie .....	176
FitNesse .....	179
Jakie linie rysować i kiedy to robić? .....	181
A co z wejściem i wyjściem? .....	183

---

Architektura wtyczek .....	184
A jednak wtyczki .....	185
Wnioski .....	187
<b>Rozdział 18. Anatomia granic .....</b>	<b>189</b>
Przekraczanie granic .....	190
Straszliwy monolit .....	190
Instalowanie komponentów .....	192
Wątki .....	193
Procesy lokalne .....	193
Usługi .....	194
Wnioski .....	194
<b>Rozdział 19. Zasady i poziomy .....</b>	<b>195</b>
Poziomy .....	196
Wnioski .....	199
<b>Rozdział 20. Reguły biznesowe .....</b>	<b>201</b>
Encje .....	202
Przypadki użycia .....	203
Modele żądania i odpowiedzi .....	205
Wnioski .....	206
<b>Rozdział 21. Krzycząca architektura .....</b>	<b>207</b>
Motyw architektury .....	208
Cel architektury .....	208
A co z siecią WWW? .....	209
Framework to narzędzie, a nie styl życia .....	209
Testowanie architektury .....	210
Wnioski .....	210
<b>Rozdział 22. Czysta architektura .....</b>	<b>211</b>
Zasada zależności .....	213
Typowy scenariusz .....	217
Wnioski .....	218
<b>Rozdział 23. Prezentery i skromne obiekty .....</b>	<b>219</b>
Wzorzec projektowy skromny obiekt .....	220
Prezentery i widoki .....	220
Testowanie i architektura .....	221
Bramy do baz danych .....	221
Mapowanie danych .....	222
Serwisy .....	222
Wnioski .....	223

---

<b>Rozdział 24. Granice częściowe .....</b>	<b>225</b>
Pomiń ostatni krok .....	226
Granice jednowymiarowe .....	227
Fasady .....	227
Wnioski .....	228
<b>Rozdział 25. Warstwy i granice .....</b>	<b>229</b>
Hunt the Wumpus .....	230
Czysta architektura? .....	231
Przekraczanie strumieni .....	234
Dzielenie strumieni .....	234
Wnioski .....	236
<b>Rozdział 26. Komponent Main .....</b>	<b>239</b>
Najważniejszy detal .....	240
Wnioski .....	243
<b>Rozdział 27. Serwisy, duże i małe .....</b>	<b>245</b>
Architektura serwisów? .....	246
Zalety serwisów? .....	246
Problem z kotkami .....	248
Pomogą nam obiekty .....	249
Serwisy bazujące na komponentach .....	251
Sprawy ogólnosystemowe .....	251
Wnioski .....	253
<b>Rozdział 28. Granice testów .....</b>	<b>255</b>
Testy jako komponenty systemu .....	256
Projekt ułatwiający testy .....	257
API testujące .....	257
Wnioski .....	259
<b>Rozdział 29. Czysta architektura osadzona .....</b>	<b>261</b>
Test n-App-stawienia .....	264
Problem docelowego sprzętu .....	266
Wnioski .....	276



---

<b>Część VI. Szczegóły .....</b>	<b>277</b>
<b>Rozdział 30. Baza danych jest szczegółem .....</b>	<b>279</b>
Relacyjne bazy danych .....	280
Dlaczego systemy baz danych są takie powszechne? .....	280
A gdyby nie było dysków? .....	282
Szczegóły .....	282
A co z wydajnością? .....	283
Anegdota .....	283
Wnioski .....	284
<b>Rozdział 31. Sieć WWW jest szczegółem .....</b>	<b>285</b>
Wieczne wahadło .....	286
Rezultat .....	288
Wnioski .....	289
<b>Rozdział 32. Frameworki są szczegółem .....</b>	<b>291</b>
Autorzy frameworków .....	292
Małżeństwo asymetryczne .....	292
Ryzyko .....	293
Rozwiązanie .....	294
Teraz ogłaszam was... .....	294
Wnioski .....	295
<b>Rozdział 33. Studium przypadku. Sprzedaż filmów .....</b>	<b>297</b>
Produkt .....	298
Analiza przypadków użycia .....	298
Architektura komponentów .....	300
Zarządzanie zależnościami .....	301
Wnioski .....	302
<b>Rozdział 34. Zaginiony rozdział .....</b>	<b>303</b>
Pakowanie w warstwy .....	304
Pakowanie według funkcji .....	306
Porty i adaptory .....	306
Pakowanie według komponentów .....	310
Diabeł tkwi w szczegółach implementacji .....	314
Organizacja a hermetyzacja .....	315
Inne sposoby rozdzielania .....	318
Wnioski. Zaginiona porada .....	319

---

<b>Dodatki .....</b>	<b>321</b>
<b>Dodatek A. Archeologia architektury .....</b>	<b>323</b>
System księgowości Union .....	324
Cięcie laserowe .....	331
Monitorowanie odlewów aluminium .....	334
4-TEL .....	335
Komputer SAC .....	340
Język C .....	344
BOSS .....	346
pCCU .....	347
DLU/DRU .....	349
VRS .....	351
Elektroniczny recepcjonista .....	353
System wysyłania serwisantów .....	355
Clear Communications .....	358
ROSE .....	360
Egzamin na architekta .....	363
Wnioski .....	365
<b>Posłowie .....</b>	<b>367</b>
<b>Skorowidz .....</b>	<b>371</b>

---

# 1 Czym są projekt i architektura?

---



W ostatnich latach pojawiło się sporo zamieszania wokół projektów i architektury. Czym właściwie jest projekt? A czym jest architektura? Czym różnią się te dwie rzeczy?

Jednym z celów tej książki jest rozwianie tych wszystkich wątpliwości i raz na zawsze zdefiniowanie, czym jest projekt, a czym architektura. Na początek muszę zapewnić, że między nimi nie ma żadnej różnicy. *Absolutnie żadnej.*

Słowo „architektura” jest często używane w kontekście czegoś wysokopoziomowego, zupełnie oderwanego od niskopoziomowych szczegółów. Z kolei „projekt” zwykle dotyczy ma struktur oraz decyzji wpływających na elementy niskopoziomowe. Ten podział nie ma jednak żadnego sensu, jeżeli przyjrzesz się rzeczom, które robi architekt systemu.

Zastanów się, co robi architekt projektujący nowy dom. Czy taki dom ma architekturę? Oczywiście, że ma. A czym dokładnie jest architektura domu? Tutaj można wymienić ogólny kształt, wygląd zewnętrzny, elewację, a także rozkład pomieszczeń. Jeżeli jednak dokładniej przyjrzesz się rysunkom sporządzonym przez architekta, zobaczysz ogromną liczbę drobnych szczegółów. Dowiesz się, gdzie zaplanowane są gniazdzka elektryczne i wyłączniki światła, a także same wyprowadzenia na oświetlenie. Zobaczysz też, który wyłącznik steruje daną lampą. Z rysunku odczytasz, gdzie znajdzie się ogrzewanie, umiejscowienie i wielkość zbiornika na ciepłą wodę oraz układ rur kanalizacyjnych. Znajdziesz tam też dokładne informacje o konstrukcji ścian, stropów i fundamentów budynku.

W skrócie — na rysunkach architekta zobaczysz wszystkie niskopoziomowe szczegóły współgrające z decyzjami wysokopoziomowymi. Okazuje się, że te niskopoziomowe szczegóły i wysokopoziomowe decyzje razem tworzą jeden projekt domu.

I podobnie jest w przypadku projektów oprogramowania. Niskopoziomowe szczegóły i wysokopoziomowa struktura stanowią razem jedną całość. Tworzą one materiał, który definiuje ogólny kształt systemu. Nie można wyodrębnić tylko jednego z tych komponentów, ponieważ nie oddziela ich żadna wyraźna linia. To jest po prostu kontinuum decyzji od tych najogólniejszych po najbardziej szczegółowe.

## Cel?

Jaki jest zatem cel tych wszystkich decyzji? Jaki jest cel tworzenia dobrych projektów oprogramowania? Tym celem nie jest nic innego jak moja definicja utopii:

*Celem architektury oprogramowania jest zminimalizowanie liczby ludzi wymaganych do zbudowania i utrzymywania danego systemu.*

Miarą jakości projektu jest po prostu miara nakładów pracy, które trzeba ponieść, żeby spełnić oczekiwania klienta. Jeżeli pracy nie będzie bardzo dużo i jej ilość nie będzie rosła w całym czasie życia systemu, to znaczy, że projekt był dobry. Jeżeli jednak każde następne wydanie oprogramowania powoduje wzrost ilości pracy, to projekt był zły. To naprawdę jest takie proste.

## Studium przypadku

W ramach przykładu zaprezentuję następujące studium przypadku. Posiłkuje się tutaj rzeczywistymi danymi z istniejącej firmy, która jednak chciałaby pozostać anonimowa.

Najpierw przyjrzyjmy się tendencjom wzrostowym w zespole inżynierów. Z pewnością każdy się zgodzi, że to bardzo optymistyczny trend. Wzrost podobny do przedstawionego na rysunku 1.1 musi wskazywać niesamowity sukces firmy!

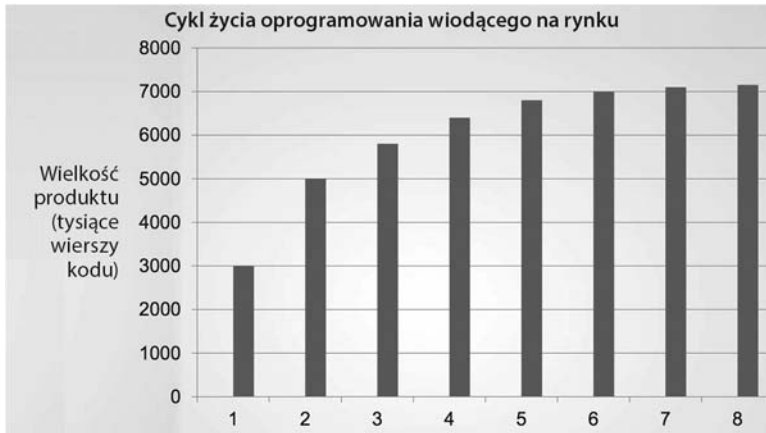


**Rysunek 1.1.** Wzrost wielkości zespołu inżynierskiego

*Reprodukcja slajdu z prezentacji Jasona Gormana, uzyskana za pozwoleniem.*

Teraz spójrzmy na wykres produktywności firmy w tym samym okresie, mierzonej po prostu jako liczba napisanych wierszy kodu (rysunek 1.2).

Coś tu się nie zgadza. Mimo że każde następne wydanie produktu obsługiwane jest przez coraz większą liczbę programistów, to jednak wygląda na to, że wielkość kodu zaczyna dążyć do jakiejś asymptoty.



**Rysunek 1.2.** Produktywność firmy w tym samym okresie

A teraz naprawdę przerażający wykres. Na rysunku 1.3 przedstawiam zmiany kosztu jednego wiersza kodu, która dokonała się w tym czasie.



**Rysunek 1.3.** Koszt wiersza kodu w zadanym okresie

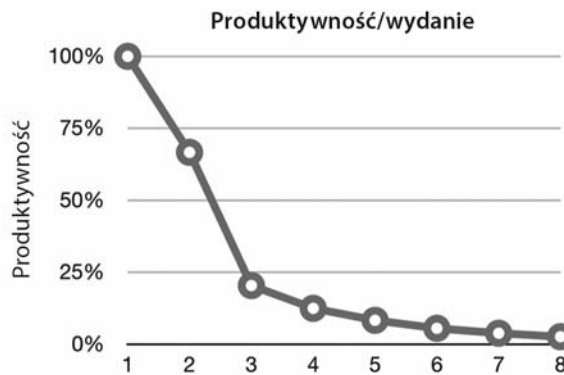
Takie trendy nie mogą się długo utrzymać. I nie ma znaczenia, jak bardzo bogata jest firma w danym momencie. Takie krzywe nieodwołalnie wydrenują cały zysk z modelu biznesowego, przez co firma przestanie się rozwijać, być może nawet zbankrutuje.

Co spowodowało tę niezwykłą zmianę produktywności? Dlaczego kod w ósmym wydaniu jest 40 razy droższy niż w pierwszym?

## Oznaki bałaganu

Masz właśnie okazję oglądać doskonale oznaki bałaganu. Jeżeli system jest składany w pośpiechu, jeżeli jedynym wyznacznikiem szybkości rozwoju jest liczba programistów, a do czytelności kodu albo jakości projektu nie przykłada się praktycznie żadnej wagi, to tę krzywą można by prześledzić aż do paskudnego końca.

Na rysunku 1.4 możesz zobaczyć, co te wszystkie wykresy oznaczają dla programistów. Zaczynali przy niemal stuprocentowej produktywności, ale z każdym kolejnym wydaniem ich produktywność spadała. Już przy czwartym wydaniu było oczywiste, że ich produktywność będzie spadać asymptotycznie do zera.



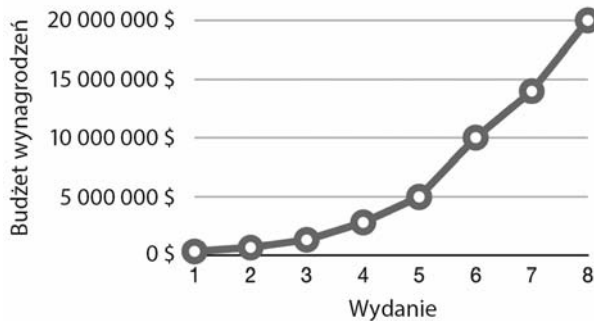
**Rysunek 1.4.** Produktywność w kolejnych wydaniach

Z punktu widzenia programistów jest to niezwykle frustrujące, ponieważ każdy z nich bardzo ciężko pracuje. Nikt w zespole nie zmniejszył swojego zaangażowania.

A mimo to, mimo heroicznym wysiłków, nadgodzin i poświęcenia nie udaje im się już zrealizować założonych zadań. Po prostu ich czas nie jest przeznaczony na tworzenie nowych funkcji, ale zjadany jest przez próby ogarnięcia całego tego bałaganu. Ich praca aktualnie polega jedynie na przenoszeniu części bałaganu z jednego miejsca w inne. I w kolejne, i w kolejne, i w jeszcze jedno. W efekcie na tworzenie nowych funkcji nie mają już praktycznie czasu.

## Okiem zarządu

Jeżeli sądzisz, że *to wszystko* wygląda fatalnie, to wyobraź sobie, jak cały ten bałagan wygląda z punktu widzenia zarządu! Przyjrzyj się rysunkowi 1.5, na którym przedstawiony został budżet wynagrodzeń programistów z tego samego okresu.



**Rysunek 1.5.** Miesięczny budżet wynagrodzeń programistów w zależności od wydania

Przy pierwszym wydaniu miesięczny budżet wynagrodzeń wynosił zaledwie kilkaset tysięcy dolarów. W drugim wydaniu zwiększył się o kolejnych kilkaset tysięcy. Jednak już ósme wydanie wymagało budżetu wynoszącego 20 milionów dolarów, który ciągle rósł.

Już tylko ten wykres jest przerażający. Od razu widać, że dzieje się coś bardzo niedobrego. Można mieć nadzieję, że przychody firmy rosną szybciej od kosztów i dzięki temu takie wydatki są usprawiedliwione. Niemniej jednak już sam kształt tej krzywej daje powody do niepokoju.

Teraz porównaj jednak krzywą z rysunku 1.5 z liczbą wierszy kodu tworzonych w każdym wydaniu oprogramowania z rysunku 1.2. Początkowe kilkaset tysięcy dolarów miesięcznie przynosiło naprawdę wiele nowych funkcji, jednak ostatnie wydatki na poziomie 20 milionów dolarów nie wносиły już prawie nic nowego do systemu. Każdy CFO, widząc te dwa wykresy, będzie wiedział, że konieczne jest wprowadzenie natychmiastowych zmian, jeżeli firma chce uniknąć katastrofy.

Tylko jakie zmiany należałoby tutaj wprowadzić? Co było robione źle? Co spowodowało ten dramatyczny spadek wydajności? Co może zrobić kadra zarządzająca oprócz oczywistego tupania i wyładowywania wściekłości na programistach?

### Gdzie szukać przyczyny?

Niemal 2600 lat temu Ezop opowiedział bajkę o żółwiu i zającu. Morał tej bajki przedstawiany był już na wiele różnych sposobów:

- „Wyścig wygrywa się powoli i miarowo”.
- „Wyścigu nie wygrywa najszybszy, a walki najsilniejszy”.
- „Spiesz się powoli”.



Sama opowieść Ezopa jest ilustracją głupoty i nadmiernej pewności siebie. Zając był tak pewny swojej niezwyklej prędkości, że nie brał wyścigu na poważnie i dlatego spał, gdy zółw przekraczał linię mety.

Dzisiejsi programiści biorą udział w podobnym wyścigu i wykazują podobny poziom pewności siebie. Oczywiście nie śpią sobie w pracy. Co to, to nie! Większość z nich tyra, na ile pozwala im zdrowie. A mimo to część ich umysłu pozostaje *uśpiona*. Dokładnie ta część, która wie, jak *ważny* jest dobry, czysty i prawidłowo zaprojektowany kod.

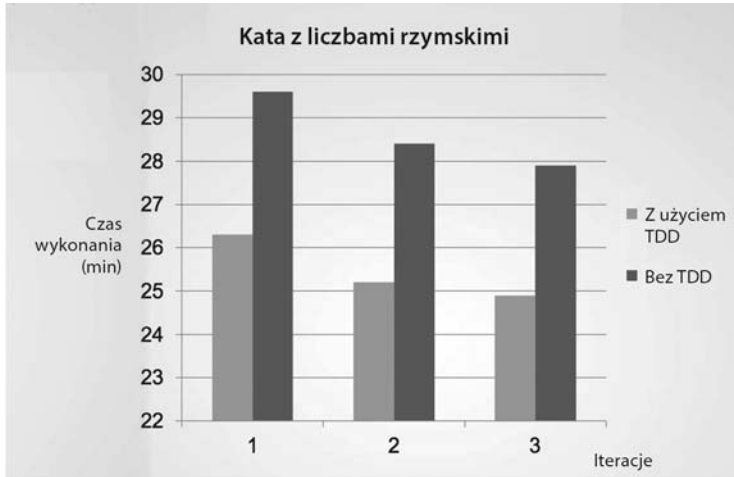
Tacy programiści poddają się bardzo powszechnemu kłamstwu: „Posprzątam tu później. Teraz musimy jak najszybciej przygotować kolejne wydanie!”. Oczywiście później nigdy nie ma już czasu na sprzątanie bałaganu w kodzie, ponieważ rynek wymaga coraz to nowszych wersji oprogramowania. Jeżeli na rynku pojawi się nowy produkt, to krótko potem powstaje cała rzesza konkurentów, przed którymi można tylko uciekać najszybciej, jak się tylko da.

I właśnie dlatego programiści już nigdy nie mają szansy niczego posprzątać. Nie mają na to czasu, ponieważ cały czas zmuszeni są do implementowania kolejnych funkcji. I jeszcze jednej. I jeszcze! W ten sposób bałagan się nawarstwia, przez co wydajność pracy zaczyna zbliżać się asymptotycznie do zera.

Podobnie jak zając zbyt wielką pewność pokładał w swojej prędkości, tak programiści łudzą się, że cały czas będą mogli zachowywać swoją wysoką wydajność pracy. Niestety, powiększający się bałagan w kodzie, który ciągle podgryza ich wydajność, nigdy nie zasypia ani nie daje wytchnienia. Jeżeli mu się poddać, to w ciągu zaledwie kilku miesięcy zmniejszy produktywność zespołu praktycznie do zera.

Inne kłamstwo, któremu dają się oszukać programiści, mówi, że tworzenie bałaganiarskiego kodu pozwala na szybszą pracę w krótkim terminie, ale na dłuższą metę zawsze ich spowalnia. Programiści akceptujący taki stan wykazują się przede wszystkim nieuzasadnioną wiarą w swoje możliwości przejścia z trybu tworzenia bałaganu w tryb sprzątania. Co więcej, w zadziwiający sposób nie chcą zauważyć, że *tworzenie bałaganu jest zawsze wolniejsze od czystej pracy*. Niezależnie od tego, o jakiej skali czasu mówimy.

Przyjrzyjmy się teraz ciekawemu eksperymentowi przeprowadzonemu przez Jasona Gormana, którego wyniki można zobaczyć na rysunku 1.6. Jason przeprowadził test trwający dokładnie sześć dni. Każdego dnia przygotowywał prosty program konwertujący liczby całkowite do zapisu rzymskiego. Określił, że praca zostanie uznana za ukończoną, jeżeli jego kod przejdzie serię przygotowanych wcześniej testów akceptacyjnych. Każdego dnia wykonanie tego zadania zajmowało nieco poniżej 30 minut. Pierwszego, trzeciego i piątego dnia Jason korzystał z dobrze znanej metody utrzymywania czystości kodu, czyli **TDD** (ang. *Test-Driven Development*). W trakcie pozostałych trzech dni pisał swój kod bez zachowywania takiej dyscypliny.



**Rysunek 1.6.** Czas wykonania zadania w kolejnych iteracjach z użyciem i bez użycia TDD

Przed wszystkim należy zauważyć poprawę czasu wykonania zadania. W kolejnych dniach prace kończyły się szybciej niż w poprzednich. Trzeba też zauważyć, że prace w dniach z TDD posuwały się mniej więcej o 10% szybciej niż w dniach, w których Jason nie stosował tej metody. Co więcej, nawet najwolniejszy dzień z TDD okazał się szybszy od najszybszego dnia bez TDD.

Część osób, patrząc na ten wykres, pomyśli zapewne, że taki wynik był nieoczekiwany. Ale dla tych, którzy nie poddali się iluzji nadmiernej pewności siebie, taki wynik nie będzie żadnym zaskoczeniem. Oni znają prawdę o dobrym tworzeniu oprogramowania:

*Jeżeli chcesz działać szybko, to działaj powoli.*

I to jest właśnie odpowiedź na dylematy zarządu. Jedynym sposobem na zatrzymanie spadku produktywności i rosnących kosztów jest zmuszenie programistów do porzucenia mentalności pewnego siebie zająca i przyjęcia odpowiedzialności za bałagan, do którego doprowadzili.

Programiści mogą pomyśleć, że odpowiedzią na te problemy powinno być rozpoczęcie prac nad systemem od nowa i zbudowanie go na podstawie innego projektu. Niestety, w ten sposób znowu do głosu dochodzi zając. Nadmierna pewność siebie, która doprowadziła do powstania tego bałaganu, podpowiada im, że mogą ten sam system zbudować znacznie lepiej, jeżeli tylko zaczną prace od nowa. Rzeczywistość jest jednak znacznie mniej optymistyczna:

*Ich przesadna pewność siebie sprawi, że nowy system ugrzęźnie w dokładnie tym samym bałaganie co pierwotny.*

## Wnioski

W każdym przypadku najlepszym rozwiązaniem dla działu oprogramowania jest przyznanie się do nadmiernej pewności siebie i podjęcie prób jego unikania. Dzięki temu będzie można przejść do poważnego traktowania jakości architektury tworzonego oprogramowania.

Jednak chcąc poważnie traktować architekturę oprogramowania, trzeba wiedzieć, czym ona właściwie jest. Chcąc tworzyć system, w którym projekt i architektura minimalizują niezbędne nakłady pracy i maksymalizują produktywność, musisz wiedzieć, które elementy architektury systemu będą prowadziły do takich właśnie wyników.

I o tym będę mówił w tej książce. Opiszę w niej wygląd dobrej i czystej architektury i projektu. Dzięki temu programiści będą mogli tworzyć systemy przynoszące zyski przez długi czas.



---

# Skorowidz

---

## A

API testowe, 258  
architektura, 28, 38, 39, 40, 43,  
154, 155, 156, 166, 208, *Patrz  
też[ 1]:* projekt  
archeologia, 324  
bazująca na komponentach, 249  
BCE, 212  
cel, 28, 208  
czysta, 231, 262, 266, 267, 270,  
272, 274  
DCI, 212  
DLU/DRU, 350  
granica, 176, 180, 181, 189, 190,  
193, 194, 231, 251  
częściowa, 226, 227  
jednowymiarowa, 227  
przekraczanie, 190, 191, 231,  
2016  
heksagonalna, 212

MVC, 215  
pakowania według  
funkcji, 306, 316  
komponentów, 312, 314, 317  
podziału pionowego, 306  
porty i adaptery, 306, 313, 315,  
316, 318  
serwisów, 246  
warstwowa, 304, 305, 310, 316  
zasady, 196, 198

## B

baza danych, 68, 91, 157, 158, 169,  
179, 180, 181, 185, 280, 282  
brama, 221  
interfejs, *Patrz:* interfejs bazy  
danych  
obiektowa, 361, 362  
relacyjna, 280, 283, 284  
UNIFY, 352

Beck Kent, 264  
biblioteka, 117, 126, 192  
ORM, 222  
STL, 294  
wstrzykiwania zależności, 240  
Booch Grady, 359, 360  
brama do bazy, *Patrz:* baza danych  
brama

## C

Carew Mike, 350  
Checkstyle, 312  
Church Alonzo, 46, 70  
Cockburn Alistair, 212  
Codd Edgar, 280  
Constantine Larry, 53  
Conwaya prawo, *Patrz:* prawo  
Conwaya  
Coplien James, 212

**D**

Dahl Ole-Johan, 46, 58  
 dane  
     baza, *Patrz:* baza danych  
     biznesowe, 202  
     mapowanie, 222  
     strumień, 233  
 Debets Maria, 50  
 debugger, 38  
 dekompozycja, 52  
     funkcyjna, 55  
 DeMarco Tom, 53  
 dependency injection, *Patrz:*  
     wstrzykiwanie zależności  
 Dependency Inversion Principle,  
     *Patrz:* reguła DIP  
 diagram Boocha, 360  
 Dijkstra Edsger Wybe, 46, 50, 51, 52  
 domknięcie, 125  
 duplikacja, 171  
 dziedziczenie, 58, 61, 91, 98, 109  
     jednobazowe, 62  
     wielobazowe, 62

**E**

Eisenhower Dwight, 40  
 encapsulation, *Patrz:*  
     hermetyzacja  
 encja, 202, 203, 214  
     przepływ danych, 214  
 event sourcing, *Patrz:* strumień  
     zdarzeń

**F**

fabryka abstrakcyjna, 109  
 Feathers Michael, 78  
 firmware, 262, 263, 267

Fragile Tests Problem, *Patrz:*  
     problem kruchych testów,  
     test kruchy  
 framework, 209, 292, 364  
     modułowy, 318  
     ryzyko, 293  
 Freeman Steve, 212  
 funkcja, 109  
     anonimowa, 70  
     close, 63  
     main, 241, 242, 243  
     map, 70  
     open, 63  
     println, 70  
     range, 70  
     read, 63  
     seek, 63  
     take, 70  
     write, 63  
     wskaźnik, 64, 190  
     zagnieżdżona, 46

**G**

Gorman Jason, 33  
 graf  
     acykliczny skierowany, 134,  
         135, 136, 310  
     cykl, 135, 136

**H**

HAL, *Patrz:* warstwa HAL  
 hardware, 38  
 Hardware Abstraction Layer,  
     *Patrz:* warstwa HAL  
 hash table, *Patrz:* tablica skrótów  
 hermetyzacja, 58, 315  
     w języku nieobiektowym, 59  
 Hibernate, 222  
 Hopper Grace, 43

Humble object, *Patrz:* obiekt  
     skromny, wzorzec skromnych  
     obiektów

**I**

independent deployability, *Patrz:*  
     instalowanie niezależne  
 independent developability, *Patrz:*  
     rozwój niezależny  
 indukcja, 51  
 inheritance, *Patrz:* dziedziczenie  
 instalowanie niezależne, 68  
 instrukcja  
     #include, 141  
     goto, 46, 47, 51, 52  
 interaktor, 91, 93  
 Interface Segregation Principle,  
     *Patrz:* reguła ISP  
 interfejs, 91  
     abstrakcyjny, 108, 109  
     adapter, 214  
     bazy danych, 181  
     graficzny, 176  
     polimorficzny, 232  
     REST, 100, 158  
     rozdzielanie, *Patrz:* reguła  
     stabilność, 108  
     użytkownika, 68, 138, 169, 181,  
         183, 184, 215, 287, 288  
     graficzny, 220, 288, 358  
     tekstowy, 230

iteracja, 51  
 iterator, 126

**J**

Jakobson Ivar, 212  
 język  
     ALGOL, 46  
     binarny, 43

C, 59, 270, 345  
 C#, 60  
 C++, 59, 62, 294, 359  
 Clojure, 70, 71  
 Fortran, 44  
 funkcyjny, 46, 70  
 Java, 60, 108, 294  
 LISP, 46, 52  
 obiektowy, 58  
 Python, 105, 108, 144  
 Ruby, 105, 108, 144  
 typowany  
   dynamicznie, 105, 108, 109,  
   144  
   statycznie, 105, 108  
 zapytań, 169

**K**

klasa, 90, 126  
   abstrakcyjna, 108  
   konstruktor, 46  
   kontenera, 126  
   String, 108  
   wywiedziona, 251  
 kod, 43  
   binarny, 172  
   zduplikowany, 171  
   źródłowy, 172, 190  
 kompilator, 43, 117  
   A0, 43  
 komponent, 116, 121, 314  
   abstrakcyjny, 144, 146, 147, 148  
   miara, 145  
   ciąg główny, 146, 148, 149  
   Main, 240, 294  
   niestabilność, 141, 142, 148  
   numer wersji, 133  
   numer wydania, 124  
   stabilność, 139, 140, 142, 144,  
   146, 147

  pomiar, 141  
   pozycyjna, 141  
   zależność, 126, 133, 134, 138,  
   139  
   wchodząca, 141  
   wychodząca, 141  
 komputer  
   COLT, 336, 340  
   Datnet 30, 324, 326, 328  
   IBM System/7, 335  
   M365, 332, 336, 340, 343  
   PDP-11/60, 344  
   PDP-8, 332  
   SAC, 336, 340, 343  
   Varian 620/f, 329  
 kontroler, 93, 215, 216

**L**

Leningen, 124  
 linking loader, *Patrz:* program  
   ładujący  
 Liskov Barbara, 79, 98  
 Liskov Substitution Principle,  
   *Patrz:* reguła LSP

**M**

macierz Eisenhowera, 40  
 maszyna stanów, 356  
 Maven, 124, 179  
 McCarthy John, 46  
 metoda, 46  
   cotygodniowej kompilacji, 132  
   naukowa, 53, 54  
 Meyer Bertrand, 78, 90  
 mikroserwis, 167, 170  
 model  
   danych, 280  
   widoku, 220, 221  
 modem, 349

moduł, 82, 90  
   DLU/DRU, 349  
 Moore'a prawo, *Patrz:* prawo  
   Moore'a  
 możliwość  
   niezależnego instalowania,  
   *Patrz:* instalowanie  
   niezależne  
   niezależnego rozwoju, *Patrz:*  
   rozwój niezależny  
 Murphy'ego prawo, *Patrz:* prawo  
   Murphy'ego  
 MySQL, 280

**N**

NDepend, 312  
 Newkirk Jim, 363  
 Nygaard Kirsten, 46, 58

**O**

obiekt skromny, 220, 221, 222  
 odwrócenie zależności, 67  
 Open-Closed Principle, *Patrz:*  
   reguła OCP  
 operating system abstraction  
   layer, *Patrz:* warstwa OSAL  
 oprogramowanie, 43, 262  
   architektura, *Patrz:*  
   architektura  
   czas życia, 262  
   miękkie, 38, 157  
   osadzone, 266, 267, 268  
   zachowanie, 38, 39, 40, 157  
 Oracle, 280  
 OSAL, *Patrz:* warstwa abstrakcji  
   systemu operacyjnego  
 OSGi, 318

## P

Page-Jones Meilir, 53  
 pamięć, 281  
   dostęp, 282  
   dyskowa, 281, 282, 325  
   EPROM, 337  
   podręczna, 281  
   RAM, 282  
   ROM, 337, 338  
   taśmowa, 332, 336  
   transakcyjna, 72, 73  
   współdzielona, 356  
 plik  
   .c, 120  
   .dll, 116, 192  
   .exe, 116  
   .gem, 116  
   .jar, 116, 121, 179, 192, 226  
   .o, 120  
   .war, 116  
   acmetypes.h, 271  
   archiwum, 116  
   implementacji, 60  
   nagłówkowy, 59, 60, 271, 276  
   stdint.h, 271  
   wykonywalny, 116  
 poczta głosowa, 354  
 podtyp, 98  
 polimorfizm, 46, 47, 58, 63, 64, 65, 66, 339  
   dynamiczny, 190, 216  
 polymorphism, *Patrz:* polimorfizm  
 poziom, 196  
 prawo  
   Conwaya, 78, 167  
   Moore'a, 120  
   Murphy'ego, 120  
 prezenter, 91, 215, 216, 218, 220, 221  
   wywołanie, 216  
 problem kruchych testów, 257

proces  
   lokalny, 193  
   uruchamiany z wiersza  
     poleceń, 194  
 procesor  
   80286, 343  
   8085, 337, 346  
   8086, 342  
 program  
   konsolidujący, 119, 120  
   ładujący, 119, 120  
   poddających się dowodzeniu, 54  
   struktura, 51  
 programowanie  
   funkcyjne, 46, 47, 70, 71  
   obiektywne, 46, 58, 60, 68, 190  
   paradygmat, 44  
   strukturalne, 46, 50, 51, 52  
     dekompozycja, *Patrz:*  
       dekompozycja  
 projekt, 28, 29, *Patrz też:*  
   architektura  
 Pryce Nata, 212  
 przerwanie programowe, 335

## R

rachunek lambda, 46, 70  
 RDBMS, *Patrz:* system  
   zarządzania relacyjnymi  
   bazami danych  
 Reenskaug Trygve, 212  
 reguła  
   acyklicznych zależności, *Patrz:*  
     reguła ADP  
   ADP, 132  
   biznesowa, 68, 79, 86, 138, 157,  
     169, 180, 181, 202, 203, 287  
   aplikacji, 169  
   aplikacyjna, 214, 222  
   CCP, 124, 125, 126, 127, 168, 310

CRP, 124, 126, 127, 128, 310  
 DIP, 79, 107, 136, 143, 216  
 DRY, 276  
 ISP, 79, 104, 105, 127  
   przykład, 104  
 istotności numeru wydania,  
   *Patrz:* reguła REP  
 jednej odpowiedzialności,  
   *Patrz:* reguła SRP  
 komponentów, 113  
 LSP, 79, 98, 99  
   naruszenie, 98, 100  
 OCP, 78, 90, 94, 95, 125  
   przykład, 90  
 odwracania zależności, *Patrz:*  
   reguła DIP  
 otwarty-zamknięty, *Patrz:*  
   reguła OCP  
 podstawiania, *Patrz:* reguła LSP  
 podziału inter, *Patrz:* reguła ISP  
 REP, 124, 127, 128, 310  
 SAP, 145  
 SDP, 139  
 SOLID, 77, 78, 82, 249, 310  
 SRP, 78, 82, 87, 91, 125, 126, 168  
   przykład, 83  
 wspólnego  
   domknięcia, *Patrz:* reguła  
     CCP  
   użycia, *Patrz:* reguła CRP  
 zależności, 213, 250, 256, 294  
   niecyklicznych, *Patrz:* reguła  
     ADP  
 relokacja, 119  
 rezystor, 331  
 rozwój niezależny, 68  
 RVM, 124



**S**

Schmidt Doug, 262  
 sekwencja, 51  
 selekcja, 51  
 serwer  
   MySQL, 179, 180  
   uucp, 359  
 serwis, 170, 222  
   bazujący na komponentach, 251  
   zalety, 246, 247, 248  
 sieć WWW, 286, 288  
   sprzedaż filmów, 298  
 Single Responsibility Principle,  
   *Patrz:* reguła SRP  
 słowo kluczowe  
   private, 60  
   protected, 60  
   public, 60  
 software, 38  
 SQL Server, 280  
 sterta, 46  
 Structure101, 312  
 strumień zdarzeń, 74  
 syndrom dnia następnego, 132  
 system  
   architektura, *Patrz:*  
     architektura  
   BOSS, 346  
   CDS, 355  
   cykl życia, 155  
     działanie, 156, 166  
     konserwacja, 157  
     wdrożenie, 155, 167  
   kontrola wersji kodu, 25  
   operacyjny, 273, 346  
   pCCU, 347  
   plików, 281  
   przypadek użycia, 166, 168, 169,  
     170, 172  
   ROSE, 361, 362

rozwój, 155, 167, 171  
 VRS, 352  
 wdrożenie, 155, 167  
 zarządzania relacyjnymi  
   bazami danych, 281

**T**

tablica skrótów, 216  
 TDD, 33  
 terminal CRT, 327, 328  
 test, 54, 221, 256  
   akceptacyjny, 179  
   jednostkowy, 210, 220  
   kruchy, 257  
   n-App-stawienia, 264, 265, 267  
   związanie, 258  
 Test-Driven Development, *Patrz:*  
   TDD  
 Turing Alan, 43, 46

**U**

urządzenie wejścia-wyjścia, 63, 65,  
 157, 159, 288  
 usługa, 194  
   restful, 100

**V**

van Wijngaarden Adriaan, 50  
 view model, *Patrz:* model widoku  
 Visual Studio, 185  
 Vogel Billy, 360

**W**

warstwa, 267  
   abstrakcji sprzętu, *Patrz:*  
     warstwa HAL  
   adapterów interfejsów, 214  
   bazy danych, 222

encji, 214  
 HAL, 268, 269, 273, 275  
 modelu, 215  
 OSAL, 274, 275  
 przekraczanie granic, 216  
 przypadków użycia, 214, 222  
 wątek, 193  
 widok, 91, 93, 215, 218, 220  
   model, *Patrz:* model widoku  
 wielowątkowość, 71  
 współbieżność, 71, 72  
 wstrzykiwanie zależności, 240, 294  
 wtyczka, 65, 184, 294, 339  
 wydajność, 283  
 wyjątek, 52  
 wykopaliska, 157  
 wyścig, 71  
 wzorzec  
   fasada, 227  
   skromnych obiektów, 220

**Y**

Yourdon Ed, 53

**Z**

zachowanie trudne do  
   przetestowania, 220  
 zakleszczenie, 71  
 zasada, *Patrz:* reguła  
 złączenie, 84, 85  
 zmienna  
   lokalna, 46  
   modyfikacja, 71  
   niezmienna, 71, 72, 73  
   obiekty, 46



# 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>

## Czysta architektura — doskonały kod!

Pierwsze linie kodu powstawały pół wieku temu. Komputery, na które tworzone te programy, w bardzo niewielkim stopniu przypominały współczesne maszyny. Niezależnie od upływu lat, postępu technologii i powstawania wymyślnych narzędzi, języków programowania czy frameworków pewne zasady tworzenia kodu pozostają niezmiennie. Są takie same jak w czasie, gdy w 1946 roku Alan Turing pisał pierwszy kod maszynowy. Respektowanie tych zasad to warunek, że uzyska się oprogramowanie o czystej architekturze, czyli poprawne strukturalnie, łatwe w utrzymaniu i rozwijaniu, a przede wszystkim działające zgodnie z oczekiwaniami.

W tej książce w sposób jasny i bardzo interesujący przedstawiono uniwersalne zasady architektury oprogramowania wraz z szeregiem wskazówek dotyczących stosowania tych reguł w praktyce. Wyczerpująco zaprezentowano tu dostępne rozwiązania i wyjaśniono, dlaczego są one tak istotne dla sukcesu przedsięwzięcia. Publikacja jest wypełniona praktycznymi rozwiązaniami problemów, z którymi musi się mierzyć wielu programistów. Szczególnie cenne są uwagi dotyczące zapobiegania częstemu problemowi, jakim jest stopniowa utrata jakości kodu w miarę postępu projektu. Ta książka obowiązkowo powinna się znaleźć w podręcznej biblioteczkę każdego architekta oprogramowania, analityka systemowego, projektanta i menedżera!

### Z książki dowiesz się:

- ▶ Do czego muszą dążyć architekci oprogramowania i w jaki sposób mogą osiągać te cele
- ▶ Jak brzmią najważniejsze zasady projektowania oprogramowania związane z adresowaniem funkcji, separacją komponentów i zarządzaniem danymi
- ▶ W jaki sposób paradygmaty oprogramowania wzmagają dyscyplinę pracy
- ▶ Co podczas tworzenia oprogramowania jest szczególnie istotne, a co jest mniej ważne
- ▶ W jaki sposób implementować optymalne struktury dla sieci WWW, baz danych, konsoli i aplikacji osadzonych

Robert C. Martin, powszechnie znany jako „Wujek Bob”, jest programistą od 1970 roku. Był jednym z autorów Manifestu Agile. Jest autorem cenionych książek, między innymi kultowego przewodnika *Czysty kod*. Napisał również dziesiątki artykułów dotyczących zasad budowy oprogramowania. Założył firmę Uncle Bob Consulting LLC oraz współzakładał (razem ze swoim systemem Micah) firmę The Clean Coders LLC. Pracował jako redaktor naczelny pisma „The C++ Report”, był pierwszym przewodniczącym Agile Alliance. Chętnie zabiera głos na konferencjach dotyczących programowania.

 <b>helion.pl</b> HELION SA ul. Kościuszki 1c 44-100 Gliwice tel.: 32 230 98 63 helion@helion.pl	<i>Sprawdź nasze szkolenia</i> <b>SZKOLENIA</b>  AKADEMIA IT & BUSINESS <a href="http://WWW.SZKOLENIA.HELION.PL">WWW.SZKOLENIA.HELION.PL</a>	<b>KOD KORZYŚCI</b> Sięgnij po więcej! ▶  ISBN 978-83-283-4225-5  9 788328 342255
<b>INFORMATYKA W NAJLEPSZYM WYDANIU</b>		Cena: 67,00 zł