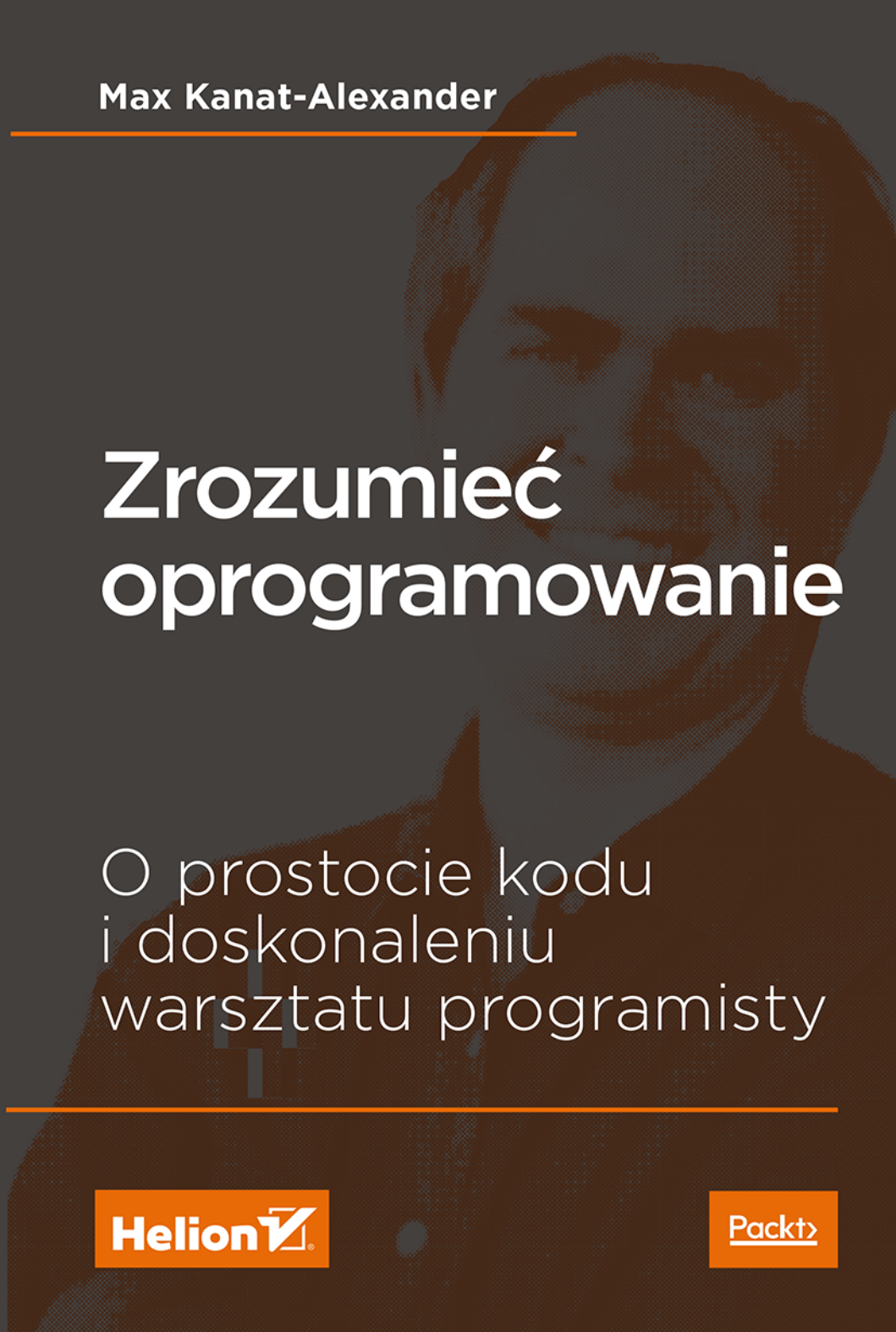




Max Kanat-Alexander

Zrozumieć oprogramowanie

O prostocie kodu
i doskonaleniu
warsztatu programisty

Helion 

Packt 

Tytuł oryginału: Understanding Software

Tłumaczenie: Radosław Małachowski

ISBN: 978-83-283-5112-7

Copyright © Packt Publishing 2017. First published in the English language under the title 'Understanding Software – (9781788628815)'

Polish edition copyright © 2019 by Helion SA

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

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

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

Autor oraz Helion SA 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 Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: 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/zroopr>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

SPIS TREŚCI

Przedmowa	11
-----------------	----

CZĘŚĆ PIERWSZA. ZASADY DLA PROGRAMISTÓW

Rozdział 1. Zanim zaczniesz... ..	17
Będziesz to robić, więc rób to dobrze	18
Rozdział 2. Postawa inżyniera	21
Rozdział 3. Niezwykła tajemnica programisty gwiazdora	25
Rozdział 4. Projekt oprogramowania w dwóch sentencjach	29

CZĘŚĆ DRUGA. ZŁOŻONOŚĆ OPROGRAMOWANIA

I JEJ PRZYCZYNY	31
Rozdział 5. Wskazówki dotyczące nadmiernej złożoności	33
Rozdział 6. Drogi do stworzenia złożoności. Zepsuj swoje API	35
Rozdział 7. Kiedy wsteczna kompatybilność nie jest warta swojej ceny?	39
Rozdział 8. Złożoność to więzienie	43

CZĘŚĆ TRZECIA. PROSTOTA I PROJEKTOWANIE OPROGRAMOWANIA	45
Rozdział 9. Projektuj od początku	47
Ruszając dobrą drogą	48
Rozdział 10. Dokładność przyszłych przewidywań	49
Rozdział 11. Prostota i precyzja	53
Rozdział 12. Dwa to za dużo	57
Refaktoryzacja	58
Rozdział 13. Rozsądny projekt oprogramowania	61
Zła droga	62
Analiza złej drogi	64
Odnosząc to do grupy	65
Dobra droga	66
Przestrzegaliśmy praw tworzenia oprogramowania	69
 CZĘŚĆ CZWARTA. DEBUGOWANIE	 71
Rozdział 14. Czym jest bug?	73
Sprzęt	74
Rozdział 15. Źródło błędów	75
Spotęgowana złożoność	76
Rozdział 16. Spraw, by to nie powróciło	79
Spraw, by to nigdy nie powróciło — przykład	80
W głąb króliczej nory	84
Rozdział 17. Fundamentalna filozofia debugowania	85
Wyjaśnij błąd	87
Patrz na system	88
Znajdź prawdziwą przyczynę	89
Cztery kroki	90

CZĘŚĆ PIĄTA. INŻYNIERIA W ZESPOŁACH	93
Rozdział 18. Efektywna produktywność inżynierii	95
Co powinienś zrobić?	97
Rozwiązanie	98
Wiarygodność i rozwiązywanie problemów	100
Blocker	101
Zmierzając w stronę podstawowego problemu	103
Rozdział 19. Mierząc produktywność dewelopera	107
Definicja „produktywności”	108
Czemu nie „linie kodu”?	108
Określając prawidłowy wskaźnik	110
A co, jeśli Twoim produktem jest kod?	111
A co z ludźmi, którzy pracują nad produktywnością deweloperów?	111
Wniosek	113
Rozdział 20. Jak radzić sobie ze złożonością kodu w firmie programistycznej	115
Krok pierwszy — lista problemów	117
Krok drugi — spotkanie	117
Krok trzeci — raport błędów	118
Krok czwarty — priorytetyzacja	119
Krok piąty — zadanie	120
Krok szósty — planowanie	121
Rozdział 21. W refaktoryzacji chodzi o funkcjonalności	123
Być efektywnym	124
Ustalając granice refaktoryzacji	127
Refaktoryzacja nie marnuje czasu, ona go oszczędza	128
Refaktoryzacja aż do jasności	128
Podsumowanie	130
Rozdział 22. Życzliwość i kodowanie	131
Oprogramowanie to ludzie	131
Przykład uprzejmości	132
Bądź miły i twórz lepsze oprogramowanie	134

Rozdział 23. Społeczność open source, w uproszczeniu	135
Utrzymanie współtwórców	136
Usuwając bariery	142
Zainteresować ludzi	145
Miej superpopularny produkt	146
Miej produkt napisany w popularnym języku programowania	146
Podsumowanie	147
 CZĘŚĆ SZÓSTA. ROZUMIEĆ OPROGRAMOWANIE	149
Rozdział 24. Czym jest komputer?	151
Rozdział 25. Komponenty oprogramowania: struktura, akcja i wynik	155
Rozdział 26. Oprogramowanie na nowo: (I)SAR wyjaśnione	157
Struktura	158
Akcja	159
Wyniki	159
ISAR w pojedynczej linii kodu	160
Podsumowując SAR	161
Rozdział 27. Oprogramowanie jako wiedza	163
Rozdział 28. Cel technologii	167
Czy są jakieś kontrprzykłady tej zasady?	168
Czy postęp technologiczny jest „dobry”?	168
Rozdział 29. Prywatność, w uproszczeniu	171
Prywatność przestrzeni	171
Prywatność informacji	173
Podsumowanie prywatności	177
Rozdział 30. Prostota i bezpieczeństwo	179
Rozdział 31. Test-Driven Development i cykl obserwacji	183
Przykłady ODA	184
Proces wytwarzania i produktywność	185
Pierwsza ODA	187

Rozdział 32. Filozofia testowania	189
Wartość testu	190
Asercje testu	190
Granice testu	191
Założenia testu	191
Projekt testu	192
Testowanie end to end	192
Testy integracyjne	194
Testy jednostkowe	195
Rzeczywistość	196
Podróbki	197
Determinizm	199
Prędkość	200
Pokrycie	202
Wniosek — ogólny cel testowania	202
 CZĘŚĆ SIÓDMA. MNIEJ DAWAĆ CIAŁA	203
Rozdział 33. Tajemnica sukcesu: mniej dawać ciała	205
Dlaczego to zadziało?	206
Rozdział 34. Jak odkryliśmy, co dawało ciała	209
Rozdział 35. Potęga „nie”	213
Rozpoznawanie złych pomysłów	215
Nie mając lepszego pomysłu	215
Wyjaśnienie, akceptacja i uprzejmość	217
Rozdział 36. Dlaczego programiści dają ciała	219
Czego się uczyć	222
Rozdział 37. Sekret szybkiego programowania: przestań myśleć	225
Zrozumienie	226
Rysowanie	227
Rozpoczynanie	228
Pomijanie kroku	229
Problemy fizyczne	229

Spis treści

To, co rozprasza	230
Zwątpienie w siebie	230
Fałszywe pomysły	231
Zastrzeżenie	231
Rozdział 38. Pycha dewelopera	233
Rozdział 39. Spójność nie oznacza jednolitości	235
Rozdział 40. Użytkownicy mają problemy, deweloperzy mają rozwiązania	237
Zaufanie i informacja	238
Problemy pochodzą od użytkowników	238
Rozdział 41. Natychmiastowa gratyfikacja = natychmiastowa porażka	241
Rozwiązania na dłuższą metę	242
Jak zniszczyć firmę tworząc oprogramowanie	243
Rozdział 42. Sukces bierze się z wykonania, nie z innowacji	245
Rozdział 43. Oprogramowanie doskonałe	247
1. Robi dokładnie to, co użytkownik mu polecił do wykonania	248
2. Zachowuje się dokładnie tak, jak użytkownik oczekuje, że się zachowa	249
3. Nie blokuje użytkownika przed komunikowaniem jego intencji	250
Doskonałość jest istotniejsza (ale nie w konflikcie) od prostoty kodu	252
Skorowidz	253

2

POSTAWA INŻYNIERA

Postawę, którą powinien przyjmować każdy inżynier dowolnej specjalizacji, można ująć w zdaniu:

Mogę rozwiązać ten problem we właściwy sposób.

Nieważne, jaki jest problem — zawsze jest *właściwy* sposób jego rozwiązania. Ten sposób może być znany i może zostać zaimplementowany. Jedynym zasadnym powodem niestosowania właściwego sposobu jest *brak zasobów*. Jednak zawsze powinieneś zakładać, że właściwy sposób istnieje, że *jesteś w stanie* rozwiązać problem we właściwy sposób i że mając odpowiednio dużo zasobów, *rozwiązałbyś* problem w odpowiedni sposób.

„Odpowiedni sposób” z reguły oznacza „sposób, który uwzględni wszystkie możliwe przyszłe sytuacje, nawet nieznane i niedające się wyobrazić”.

Most, który znosi bez uszczerbku wszelkie możliwe warunki środowiskowe czy dające się przewidzieć natężenie ruchu bez ciągłych prac konserwacyjnych, został zbudowany „we właściwy sposób”.

Kod, który zachowuje swoją prostotę, jednocześnie zapewniając elastyczność potrzebną przy możliwych przyszłych ulepszeniach, jest zaprojektowany „we właściwy sposób”.

Jest wiele bezzasadnych przyczyn rozwiązywania problemu w niewłaściwy sposób:

- **Nie znam właściwego sposobu.** Często znalezienie właściwego sposobu wymaga lepszego zrozumienia problemu lub doksztalcenia się. Kiedy jestem w takiej sytuacji, na chwilę odchodzę od problemu i często wpadam na pomysł rozwiązania, kiedy po prostu gdzieś idę lub następnego dnia, kiedy znowu do niego siadam. Staram się nie akceptować czegoś, co jest niewłaściwe tylko dlatego, że jeszcze nie znam właściwego sposobu.
- **Grupa nie może się zdecydować, co będzie właściwym sposobem.** Czasami grupa osób spiera się na temat tego, co będzie „właściwym sposobem”, i zagadnienie staje się zagniatwane. Grupy nie są zbyt dobre w podejmowaniu decyzji. Jak wszyscy wiemy, nie projektujesz oprogramowania w komitecie, i podejrzewam, że „projekty komitetu” w innych dziedzinach inżynierii są równie złe.

Rozwiązaniem byłoby wybranie doświadczonego i zaufanego inżyniera, który będzie rozumiał podstawowe zasady obszaru, którym się zajmujesz, i potrafi określić właściwy sposób sam, po uważnym przestudiowaniu istniejących argumentów i zebraniu odpowiednich informacji, zgodnie ze standardem i aktualnymi procedurami.

- **Jestem zbyt leniwy (zmęczony, głodny, rozkojarzony), żeby zrobić to teraz we właściwy sposób.** To zdarza się każdemu od czasu do czasu. Jest pierwsza w nocy, od 15 godzin bez przerwy pracujesz nad projektem i chcesz,

by ta jedna przeklęta rzecz zadziałała, natychmiast!
Odpocznij, pomyśl, wróć do tego później. Świat się nie skończy, a problem będzie istniał nadal i będzie możliwy do rozwiązania.

Idź spać, wyjdź coś zjeść, pójdz na spacer — zrób, co tylko możesz, by przejść na poziom mentalny, w którym możesz rozwiązać problem we właściwy sposób, i dopiero wtedy wróć. Jeżeli jesteś w stanie, w którym nie możesz rozwiązać problemu we właściwy sposób, to jest to odpowiedni czas na przerwę.

Nie zaniedbujesz swoich obowiązków, jeśli tak postępujesz — w zasadzie dobrze rozumiesz odpowiedzialność za sukces projektu, gdy mówisz: „To musi być zrobione dobrze i by zrobić to dobrze, właśnie teraz należy zrobić sobie przerwę i wrócić do tego później”.

W większości przypadków wymaga to stałej i niezmiennej wiary w siebie, wiary, że możesz rozwiązać problem we właściwy sposób.

Max

SKOROWIDZ

A

akceptacja, 217
akcja, 155, 159
API, 35
asercje testu, 190

B

bezpieczeństwo oprogramowania,
179
biblioteka, 57
blocker, 101
błędy, 73, 75
wyjaśnianie, 87
brak postępu, 40
bug, 73

C

cel technologii, 167
cykl ODA, 184

D

debugowanie, 71, 85, 86
cztery główne kroki, 90
systemu, 88

determinizm, 199
dodawanie funkcjonalności, 48
dokładność, 49
przewidywania, 49
dokumentacja, 143, 144
doskonałość, 252

E

efektywna produktywność inżynierii,
95
efektywność, 124

F

fałszywe pomysły, 231
faza obserwacji, 186
firewall, 180
firma, 243
frontendowy deweloper, 110
fundamentalne bezpieczeństwo, 180
funkcjonalności, 41, 48, 74, 125

G

granice
refaktoryzacji, 127
testu, 191

I

innowacje, 238, 245
interfejs użytkownika, 54
inżynier produktywności, 100
inżynieria w zespołach, 93
ISAR, 160

J

jednolitość, 235

K

kanały komunikacji, 143
kod jako produkt, 111
kodowanie, 131
kompatybilność wsteczna, 39
komponenty oprogramowania, 155
komputer, 151
komunikat
 o błędach, 88
 w logu, 88

L

linie kodu, 108
logi, 88

Ł

łatwe projekty, 142

M

metryka, 110
mierzenie
 linii kodu, 109
 produktywności dewelopera, 108
model-widok-kontroler, 155
MVC, Model View Controller, 155

N

nadmierna złożoność, 33
nauka, 222
negatywne nastawienie, 140

O

ODA, 184
 Obserwacja, Decyzja, Akcja, 184
 pierwsza, 187
 proces wytwarzania, 185
 produktywność, 185
odmowa, 214, 217
odpowiadanie współtwórcom, 138
odpowiedzialność, 77
odrzućcie kompatybilności
 wstecznej, 41
oprogramowanie
 doskonałe, 247
 jako wiedza, 163

P

planowanie, 121
pokora, 234
popularność produktu, 146
popularny język programowania, 146
porażka, 98, 241
postawa inżyniera, 21
postęp technologiczny, 168
prawa tworzenia oprogramowania, 68
prawidłowa metryka, 110
precyzja, 53, 56
prędkość, 200
priorytetyzacja, 119
problemy
 fizyczne, 229
 ignorowanie, 81
 nieistniejące, 104
 podstawowe, 103, 212
 rozwiązywanie, 21, 79–81, 100

- użytkowników, 237
- z wydajnością, 209
- zapobieganie, 79
- złożonego kodu, 115
- znajdowanie przyczyny, 81, 89
- zrozumienie przyczyny, 86
- proces wytwarzania, 185
- produktywność, 108, 185
 - dewelopera, 107, 111
 - inżynierii, 95
- programista, 219
 - backendowy, 110
 - doskonały, 223
 - komputerowy, 108
- projekt
 - Bugzilla, 205
 - testu, 192
- projektowanie oprogramowania, 45, 61, 220
 - od początku, 47
 - analiza, 64
 - dobry sposób, 66
 - prawa, 68
 - specyfikacja, 65
 - standardy, 66
 - zły sposób, 62
- projekty open source, 145
- prostota, 45, 53, 179, 181
- prywatność, 171, 177
 - informacji, 173
 - przestrzeni, 171
- przewidywanie, 49
- przyczyna problemu, 89
- pułapka dla programistów, 235
- pycha dewelopera, 233

R

- raport błędów, 118
- refaktoryzacja, 58, 103, 123, 125
 - oszczędność czasu, 128
 - ustalanie granic, 127
 - wzorzec, 128

- rotacja, 138
- rozpoczynanie, 228
 - złych pomysłów, 215
- rozumienie zadań, 25
- rozwiązania na dłuższą metę, 242
- rozwiązywanie problemów, 21, 79–81, 100
 - użytkowników, 237
- rysowanie, 227

S

- SAR, Structure, Action and Results, 157, 161
- specyfikacja, 65
- społeczność open source, 135
- spójność, 235
- sprzęt, 74
- struktura, akcja i wynik, 155, 158
- sukces, 208, 245
 - przeciętny, 98
- szybkie
 - programowanie, 225
 - pomijanie kroku, 229
 - problemy fizyczne, 229
 - rozpoczynanie, 228
 - rozpraszające środowisko, 230
 - rysowanie, 227
 - zrozumienie, 226
 - rozwiązanie problemu, 81

Ś

- śledzenie błędów, 205

T

- TDD, Test-Driven Development, 183
- technologia, 167
- testowanie
 - end to end, 192
 - oprogramowania, 189

testy

- asercje, 190
- cel, 202
- determinizm, 199
- granice, 191
- integracyjne, 194
- jednostkowe, 195
- podróbki, 197
- pokrycie, 202
- prędkość, 200
- projekt, 192
- skala, 196
- wartość, 190
- założenia, 191

tworzenie

- interfejsów użytkownika, 54
- złego kodu, 219
- złożoności, 35

U

- uproszczenie, 135
- uprzejmość, 132, 217
- usuwanie barier, 142
- utrzymanie współtwórców, 136

W

- wartość testu, 190
- wdrażanie złych pomysłów, 214
- wiarygodność, 100
- wiedza, 163
- wklejanie kodu, 57
- wsteczna kompatybilność, 39
- wydajność, 209
- wyjaśnienie, 217
- wyniki, 155, 159

Z

- zadanie, 120
- założenia testu, 191
- zapobieganie problemom, 79
- zasada jednej odpowiedzialności, 59
- zasady projektowania
 - oprogramowania, 29
- zastrzeżenie, 231
- zaufanie, 238
- złe pomysły, 214
 - rozpoznawanie, 215
- złoty środek, 98
- złożoność oprogramowania, 31, 43, 116
 - lista problemów, 117
 - planowanie, 121
 - priorytetyzacja, 119
 - raport błędów, 118
 - spotęgowana, 76
 - spotkanie, 117
 - zadanie, 120
- zły kod, 219
- zniszczenie firmy, 243
- zrozumienie przyczyny problemu, 86, 226
- zwątpienie, 230

Ż

- źródło błędów, 75

Ż

- życzliwość, 131

PROGRAM PARTNERSKI

— GRUPY HELION —

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

Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA
Helion 

Złożoność jest głupia. Prostota jest mądra.

W codziennej praktyce programiści często muszą sobie radzić z coraz większą złożonością tworzonego kodu. Mnożą się błędy, a ryzyko spektakularnej porażki rośnie. Jeśli ten scenariusz dotyczy także Ciebie, poziom złożoności Twoich projektów Cię przytłacza i czujesz, że nie dajesz rady, już teraz wdróż fundamentalne reguły, dzięki którym Twój kod odzyska prostotę i niezawodność! Jeśli każdy dzień z nowym projektem Cię rozczarowuje i przestałeś wierzyć w swoje możliwości, sięgnij po tę książkę!

Max Kanat-Alexander, odpowiedzialny za dział Code Health w Google, w tym świetnym zbiorze esejów dzieli się mnóstwem trafnych spostrzeżeń dotyczących zasad programowania, złożoności kodu, pracy zespołowej i filozofii projektowania aplikacji. W książce pokazano bardzo różnorodne zagadnienia, od pisania prostego kodu aż po jego debugowanie i pogłębioną analizę. Dowiesz się także, jaka postawa charakteryzuje naprawcę dobrego programistę. Świeżość spojrzenia i znakomity styl autora sprawia, że tę książkę przeczytasz z przyjemnością. Wiedza techniczna i poziom ekspertyzy przyniosły mu status guru kodu, a jego pomysły zainspirują Cię i odświeżą Twoje podejście do wyzwań związanych z byciem deweloperem. Odzyskaj radość ze swojej pracy i zapomnij o porażkach!

Ta książka skłoni Cię do przemyśleń:

- dlaczego prostota i precyzja są najważniejsze w tworzeniu kodu
- czym jest mistrzowskie programowanie
- jak poradzić sobie ze złożonością oprogramowania
- skąd się biorą porażki programistów i jak ich unikać
- jak się ma prostota do bezpieczeństwa aplikacji
- czym są błędy i jak należy rozumieć debugowanie

Max Kanat-Alexander zaczął naprawiać komputery jako ośmiolatek, a jako nastolatek był już całkiem sprawnym programistą. Jako główny architekt pracował nad projektem Bugzilla, obecnie, jako technical lead, odpowiada za dział Code Health w Google. Jest też autorem *codesimplicity.com* i *fedorafaq.org*. W społeczności programistów jest uważany za guru kodowania — dzięki imponującej wiedzy, wnikliwości i wielkiej życzliwości dla innych. Obecnie mieszka w północnej Kalifornii.

Helion helion.pl 0 801 339900 0 601 339900	<i>Sprawdź nasze szkolenia!</i> SZKOLENIA AKADEMIA IT & BUSINESS WWW.SZKOLENIA.HELION.PL	KOD KORZYŚCI Sięgnij po więcej! ▶ ISBN 978-83-283-5112-7 9 788328 351127	
INFORMATYKA W NAJLEPSZYM WYDANIU			Cena: 49,00 zł

Packt