

**Wstęp**

*Doskonałe metody tworzenia oprogramowania dzieli przepaść od metod przeciętnych. Jest to przepaść większa niż w wypadku innych dziedzin tworzenia czegokolwiek. Ważne jest więc narzędzie, które przyczyniałoby się do rozpowszechniania dobrych metod.*

Fred Brooks

Głównym powodem napisania tej książki jest fakt istnienia bariery między wiedzą, którą dysponują profesjonalści, a metodami pracy znajdującymi się w powszechnym użytku. Wiele użytecznych metod programowania jest ukrytych w czasopismach i skryptach akademickich i dopiero po wielu latach zostają zauważone i wykorzystane.

Mimo że metody tworzenia oprogramowania rozwijały się w ostatnich latach niezmiernie szybko, rozwój ten nie dokonał się w metodach stosowanych powszechnie. Wiele programów wciąż zawiera błędy, jest opóźnionych w stosunku do bieżącego stanu wiedzy, a podczas ich tworzenia nie uwzględniono wszystkich wymagań. Ponadto wiele programów nie spełnia oczekiwań użytkowników. Zarówno zawodowi programiści, jak i naukowcy odkryli efektywne metody pozwalające na uniknięcie większości problemów związanych z programowaniem, z którymi się borykano w latach 70. i 80. Jednak ponieważ te metody są zwykle publikowane tylko w wysoce specjalistycznych czasopismach technicznych, nawet w latach 90. metody te pozostawały nieznanne w wielu firmach informatycznych. Sridhar Raghavan i Donald Chand zauważyli, że od momentu odkrycia czegoś do momentu wejścia tego w życie mija zwykle od 5 do 15 lat [255]. Ta książka powinna skrócić ten proces, już teraz przybliżając główne odkrycia typowym programistom.

## **Do kogo jest kierowana ta książka**

Wyniki badań i doświadczeń programistycznych zebrane w tej książce ułatwią tworzenie dobrego jakościowo oprogramowania oraz przyspieszą pracę i pozwolą uniknąć popełniania wielu błędów. Książka umożliwi zrozumienie, dlaczego w przeszłości podczas programowania natykaliśmy się na pewne problemy oraz ukazuje, jak należy ich unikać. Opisane w niej metody programowania ułatwiają sprawowanie kontroli nad dużymi projektami oraz umożliwiają poprawianie i modyfikowanie oprogramowania, w miarę jak zmieniają się wymagania.

## **Doświadczeni programiści**

„Programista doskonały” posłuży doświadczonym programistom, którzy potrzebują wszechstronnej i łatwej w użyciu instrukcji dotyczącej konstruowania oprogramowania. Ponieważ w tej książce skoncentrowano się tu na implementowaniu, czyli na tym etapie cyklu powstawania oprogramowania, który jest doskonale znany wszystkim programistom, przedstawione tu użyteczne metody tworzenia oprogramowania będą zrozumiałe zarówno dla programistów-samouków, jak i dla tych mających już podbudowę teoretyczną.

## **Programiści-samoucy**

Osoby, które nie uczestniczyły w żadnych kursach programowania stanowią większość. Każdego roku przybywa około 100 tysięcy nowych programistów [51], ale corocznie jest nadawanych tylko 40 tysięcy tytułów naukowych z dziedziny informatyki [230]. Od razu

narzucą się więc wnioski, że umiejętności większości programistów nie są poparte żadną formalną podbudową. Wielu programistów-samouków wchodzi w skład grupy profesjonalistów — inżynierów, księgowych, nauczycieli, naukowców oraz właścicieli małych firm — w których pracy jest niezbędne programowanie, ale którzy nie myślą o sobie jako o programistach. Bez względu na zakres szkoleń odbytych przez programistów, ta książka oferuje im dostęp do efektywnych metod programowania.

## Studenci

Przeciwieństwem programisty dysponującego doświadczeniem, który jednak nie przeszedł żadnego szkolenia, jest młody absolwent uczelni. Obecni absolwenci mają często dużą wiedzę teoretyczną, ale niewielkie doświadczenie praktyczne w tworzeniu programów. Teoretyczna znajomość dobrych metod programowania jest często powolnie przekształcana na potrzeby zastosowań używanych przez wąską grupę architektów, analityków, projektantów i doświadczonych programistów. Jeszcze częściej jest ona wynikiem prób i błędów pojedynczych programistów. Ta książka stanowi alternatywę dla powolnego procesu nabywania umiejętności z wąskiej dziedziny. Zebrano w niej pomocne wskazówki i efektywne metody tworzenia oprogramowania, które wcześniej były zdobywane wyłącznie dzięki samodzielnej pracy i korzystaniu z doświadczeń innych osób. Książka upraszcza zadanie studentom, ponieważ ułatwia przejście ze środowiska akademickiego do zawodowego.

## Główne korzyści płynące z lektury tej książki

Bez względu na poziom naszej wiedzy, „Programista doskonały” ułatwi nam pisanie lepszych programów w krótszym czasie, pozwalając przy tym na uniknięcie wielu błędów.

**Kompletny podręcznik tworzenia oprogramowania.** W książce opisano ogólne zagadnienia dotyczące konstruowania, takie jak jakość oprogramowania, czy sposoby pojmowania programowania. Podano też wiele informacji szczegółowych, takich jak czynności wykonywane podczas budowania podprogramów, sposoby używania danych i konstrukcji sterujących, metody wykrywania błędów, a także metody i strategie dopracowywania kodu źródłowego. Nie trzeba czytać tej książki w całości, aby się zapoznać z tymi informacjami. Została bowiem napisana w taki sposób, aby ułatwić wyszukiwanie potrzebnych danych.

**Pytania kontrolne.** W książce zamieszczono pytania kontrolne, których można używać podczas tworzenia architektury i projektu programu, analizowania jakości modułów i podprogramów, wybierania nazw zmiennych oraz sprawdzania poprawności użytych konstrukcji sterujących, zastosowanego układu kodu, wprowadzonych mechanizmów do testowania oraz innych elementów.

**Najlepsze informacje.** Przedstawiono tu wiele najnowszych, dostępnych metod programowania, z których wiele nie weszło jeszcze do powszechnego użytku. Ponieważ informacje zawarte w książce są wynikiem nabytych doświadczeń i przeprowadzonych badań, przedstawione w niej metody będą aktualne jeszcze przez wiele lat.

**Szersza perspektywa postrzegania zagadnień tworzenia oprogramowania.** W książce dano możliwość wyjścia poza prymitywną metodę pracy, polegającą na przygotowywaniu czegoś i sprawdzaniu, czy to działa. Bardzo niewielu programistów może znaleźć dość czasu na przeczytanie dziesiątek książek informatycznych oraz setek publikacji prasowych, których główne idee zebrano w tej książce. Wyniki badań i rzeczywistych doświadczeń umieszczone w książce ukierunkują i uformują sposób myślenia o projektach, ułatwiając zaplanowanie strategii działania wykraczającego poza wykonywanie naprędce różnych czynności.

**Rozwiązania pasujące do dowolnego języka proceduralnego.** Opisano tu metody, które pozwalają na wydobycie wszystkich zalet używanego języka programowania, bez względu na to, czy jest to Pascal, C, C++, Ada, Basic, Fortran, COBOL, czy inny język.

**Przykłady kodu.** W książce zamieszczono ponad 500 przykładów dobrze i źle napisanego kodu źródłowego. Tak duża liczba zaprezentowanych przykładów wynika z tego, że najlepiej jest się uczyć właśnie na przykładach. Zasada ta obowiązuje w wypadku większości programistów.

Przykłady kodu napisano w różnych językach programowania, ponieważ opanowanie kilku języków jest zwykle barierą, którą trzeba pokonać na drodze do posady profesjonalnego programisty. Gdy programista zrozumie, że zasady programowania są związane ze składnią konkretnego języka programowania, dopiero wówczas stają przed nim otworem drzwi do zagadnień, które mają rzeczywisty wpływ na jakość i wydajność działania programów.

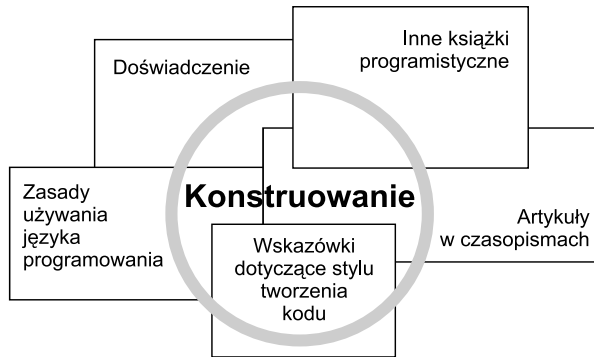
Aby ułatwić przyswojenie sobie zagadnień ginących w gąszczu wielu języków programowania, starałem się unikać opisywania elementów typowych dla konkretnego języka, za wyjątkiem miejsc, gdzie te elementy zostały specjalnie omówione. Nie trzeba rozumieć wszystkich niuansów w podanych przykładach, aby zrozumieć główne idee, które starałem się przekazać. Koncentrując się na przedstawianym problemie odkryjemy, że możemy odczytać kod programu, bez względu na to, w jakim języku został napisany. Aby jeszcze bardziej ułatwić analizowanie przykładów, wszystkie ważniejsze fragmenty zostały odpowiednio omówione.

**Odnosniki do innych źródeł informacji.** W książce zebrano wiele informacji na temat konstruowania oprogramowania, ale nie można uznać, że zostały one wyczerpane. W podrozdziałach „Warto przeczytać” podano tytuły innych książek i artykułów, które warto przeczytać, aby uzupełnić swoją wiedzę na temat zagadnień, które wydały się nam najbardziej interesujące.

## Inne miejsca, w których można znaleźć te informacje

„Programista doskonały” omawia metody konstruowania oprogramowania zabrane z wielu różnych źródeł. Nie dość, że informacje o tych metodach są bardzo rozproszone, wiele z nich nie było nawet opublikowanych [150]. Nie ma nic tajemniczego w efektywnych i wydajnych metodach programowania używanych przez doświadczonych programistów. W pośpiechu przygotowywania projektów z dnia na dzień bardzo niewielu ekspertów znajduje czas na podzielenie się doświadczeniami z innymi osobami. W związku z tym wielu programistów może mieć problemy ze znalezieniem dobrego źródła informacji.

Metody opisane w tej książce wypełniają pustkę między radami dla początkujących a publikacjami dla doświadczonych programistów. Po przeczytaniu książek „Wprowadzenie do języka C”, „Język C” oraz „Język C dla zaawansowanych” trudno znaleźć książkę, która pozwoli na dalsze poszerzenie wiedzy. Można oczywiście czytać książki poświęcone szczegółowym zagadnieniom związanym ze sprzętem i systemami operacyjnymi komputerów osobistych, komputerów Macintosh i systemów uniksowych albo związanych z konkretnymi językami programowania (bo nie można programować w jakimś środowisku nie znając dobrze tych szczegółów). Ta książka jest natomiast jedną z niewielu, w których opisano programowanie w sposób uogólniony. Najwięcej korzyści można wynieść z metod, których można używać bez względu na to, w jakim środowisku pracujemy lub jakiego języka programowania używamy. W innych książkach metody te są zwykle lekceważone i właśnie dlatego się tu na nich skoncentrowano.



Informacje przedstawione w tej książce zostały zaczerpnięte z wielu różnych źródeł

Jedyną inną drogą do uzyskania informacji, które można znaleźć w tej książce jest przeczytanie sterty książek i setek czasopism technicznych, a następnie uzupełnienie tego bogatym doświadczeniem z dziedziny programowania. Jednak nawet wtedy można wynieść wiele korzyści z tej książki, ponieważ zebrano w niej wszystkie te wiadomości, a ponadto przedstawiono je w taki sposób, aby można je było łatwo odszukać.

## Powód napisania tej książki

W środowisku programistów jest zauważalny niedobór podręczników, w których zaprezentowano by informacje na temat efektywnych metod tworzenia programów. W raporcie organizacji *Computer Science and Technology Board* stwierdzono, że największy zysk jakości i wydajności kodu można osiągnąć w drodze standaryzowania, ujednocnienia i rozpowszechniania wiedzy na temat efektywnych metod tworzenia oprogramowania [84]. W podsumowaniu wykazano, że u podstaw strategii rozpowszechniania tej wiedzy powinny leżeć podręczniki tworzenia oprogramowania.

Sama historia programowania podaje wiele konkretnych przykładów, jak bardzo są potrzebne podręczniki konstruowania oprogramowania.

## Lekceważenie zagadnienia konstruowania

W pewnym okresie opracowywanie oprogramowania i pisanie kodu było traktowane jako jedność. Jednak w miarę jak wskazywano coraz więcej różnych czynności wykonywanych w tym procesie, wielu specjalistów zaczęło się głowić i debatować nad metodami zarządzania projektem, analizowania wymagań, projektowania oraz testowania. W wyniku zidentyfikowania tych nowych obszarów, konstruowanie kodu zaczęło traktować jako mało znaczący element przygotowywania oprogramowania.

Ignorowanie konstruowania kodu pogłębiło się jeszcze bardziej, gdy zauważono, że tworzenie kodu jest najmniej przyjemnym zadaniem w procesie przygotowywania programu. Typowy programista w dużej firmie zajmuje się zwykle tworzeniem kodu podprogramów, których specyfikacje i projekt zostały przygotowane przez innego programistę, który stoi wyżej w hierarchii stanowisk w tej firmie. Po kilku latach programista zostaje awansowany i zajmuje się analizowaniem wymagań lub zarządzaniem projektem, a po kolejnych kilku latach może już z dumą przyznać, że całkowicie ma już za sobą okres, kiedy musiał pisać kod programów.

## Duże znaczenie konstruowania

Innym powodem, dla którego konstruowanie było lekceważone przez informatyków i programistów jest błędne przekonanie, że w porównaniu z innymi czynnościami wykonywanymi podczas przygotowywania oprogramowania, konstruowanie jest procesem niemalże mechanicznym i nie daje dużej swobody udoskonalania go. Nic bardziej mylnego.

Konstruowaniu zwykle trzeba poświęcić 80% wysiłku wkładanego w cały proces tworzenia małych projektów i 50% wysiłku wkładanego w przygotowywanie dużych projektów. Z procesem konstruowania jest związanych 75% błędów występujących w małych projektach i od 50% do 75% błędów występujących w średnich i dużych projektach. Wszystko, z czym jest związane od 50% do 75% błędów wymaga udoskonalenia.

Pojawiają się głosy, że chociaż błędy powstałe podczas konstruowania stanowią procentowo dużą część wszystkich błędów, to ich usuwanie jest tańsze niż usuwanie błędów powstałych podczas analizowania i projektowania, a w związku z tym są one mniej znaczące. Przekonanie, że usuwanie błędów powstałych podczas konstruowania jest stosunkowo tanie jest słuszne, ale mylące, ponieważ koszt nieusunięcia tych błędów może się okazać niezwykle wysoki. Gerald Weinberg zauważył, że trzy najkosztowniejsze błędy w historii kosztowały każdorazowo setki milionów dolarów, a były to błędy mieszczące się w jednym wierszu i zostały popełnione podczas tworzenia kodu [314]. Usunięcie tych błędów być może byłoby mniej kosztowne niż usunięcie błędów popełnionych podczas analizowania lub opracowywania architektury, ale niskie koszty usuwania błędów wcale nie oznaczają, że ich usuwanie ma mieć niski priorytet!

Jak na ironię, mimo lekceważenia czynności konstruowania, to właśnie ono jest jedynym zadaniem, które zostanie na pewno ukończone w całości. Wymagania można przyjąć, zamiast je dokładnie przeanalizować; architekturę można opracować w sposób niepełny, zamiast ją dokładnie zaprojektować; testowanie systemu można skrócić lub pominąć, zamiast w pełni zaplanować i przeprowadzić. Ale aby utworzyć gotowy program, należy go skonstruować, a to sprawia, że konstruowanie jest obszarem niezwykle odpowiednim do wprowadzania w nim udoskonaleń.

## Nie ma innej porównywalnej książki

W świetle tego, że znaczenie konstruowania jest oczywiste, byłem pewny, że ktoś napisze książkę poświęconą efektywnym metodom konstruowania. Potrzeba napisania podręcznika programowania w efektywny sposób wydawała się naturalna. Okazało się jednak, że istnieje tylko kilka książek poświęconych konstruowaniu, a w dodatku opisano w nich tylko część tego zagadnienia. Niektóre zostały napisane wiele lat temu i dotyczyły dosyć egzotycznych języków programowania, takich jak ALGOL, PL/I lub Ratfor. Autorami niektórych z tych książek byli wykładowcy, którzy nie mieli kontaktu z kodem rzeczywistych programów. Pisali o metodach przydatnych w tworzeniu projektów uczelnianych, ale mieli niewielkie pojęcie o tym, jak należy te metody przenieść do środowiska tworzenia programów używanych na szeroką skalę. W jeszcze innych książkach autorzy przedstawiali nowe metody, ale pomijali fakt istnienia ogromnych zbiorów starych metod o dowiedzionej przydatności.

Krótko mówiąc, nie mogłem znaleźć żadnej książki, w której spróbowano by chociażby przedstawić zarys praktycznych metod będących wynikiem zebranych doświadczeń, przeprowadzonych badań i prac akademickich. Potrzebne były informacje o nowych językach programowania, środowiskach interakcyjnych oraz najlepszych metodach programowania. Wydawało się, że podręcznik programowania powinien zostać napisany przez kogoś, kto doskonale opanował programowanie, zarówno na poziomie teoretycznym, jak i praktycz-

nym. Wyobrażałem sobie tę książkę jako zapis wyczerpującej dyskusji na temat konstruowania kodu, prowadzonej przez wielu programistów.

## Podziękowania

Na początku chciałbym podziękować ciężko pracującemu zespołowi wydawnictwa Microsoft Press za wysiłek, jaki włożył w realizację tego wymagającego zadania. Moje szczególne podziękowania niech przyjmą: Erin O'Connor, Mike Halvorson, Arlene Myers, Peggy Herman, Jeff Carey, Alice Smith, Lisa Theobald, Jennifer Harris, Zaafar Hasnain, Pat Forgetting, Shawn Peck, Judith Bloch, Eric Stroo, Barbara Runyan, Steve Murray, Katherine Erickson, Jeannie McGivern, Lisa Sandburg, Connie Little, Margarine Hargrave, Sally Brunsmann, Carol Luke, Ruth Pettis, Kim Eggleston, Dean Holmes, Jennifer Vick oraz Wallis Bolz. Jestem także wdzięczny pracownikom biblioteki Bellevue Lake Hills Public Library oraz Microsoft Library za wyszukiwanie dla mnie setek trudnych do znalezienia książek i artykułów.

Na powstanie tej książki miało szczególny wpływ kilku programistów, którym jestem również niezmiernie wdzięczny. Są to Barry Boehm, Capers Jones, Gerald Weinberg, Tom Gilb, Harlan Mills, David Card, Frank McGarry, Robert Grady, Bill Curtis, Ben Shneiderman, Elliot Soloway oraz Victor Basili. Należy również wspomnieć o wielu innych osobach zajmujących się tą dziedziną, ale ci, których wymieniłem, gromadzili mozolnie dane z rzeczywistych projektów, aby dostarczyć konkretną odpowiedź na pytanie, co pozwala na efektywne tworzenie oprogramowania. To dzięki pracy tych osób można całkowicie zmienić sposób działania z metody prób i błędów na metodę konsekwentnego tworzenia oprogramowania.

Podczas pisania „Programisty doskonałego” cieszyłem się szczególnie z rad dawanych mi przez grupę niezwykle ciekawych ludzi sprawdzających moją pracę. Każda z tych osób służyła mi radą w dziedzinie, którą znała tak dobrze, że mogła o niej dyskutować. Mogłem więc korzystać z wielu najlepszych źródeł. Rady, których mi udzielono wpłynęły niezmiernie korzystnie na kształt tej książki oraz pomogły mi jako programiście.

Dziękuję wszystkim osobom, które sprawdzały poszczególne rozdziały na wczesnym etapie ich powstawania. Są to: Tammy Forman, Bill Kiestler, Mike Klein, Margot Page, Peter Pathe, Jack Woolley, Joey Wyrick i Mike Zevenbergen.

O wiele więcej osób przeglądało wersję roboczą tej książki od początku do końca. Dziękuję Tony'emu Piscullemu za komentarze dotyczące układu treści książki, Dave'owi Moore'owi za dyskusowanie o tworzeniu oprogramowania w firmie Microsoft, Pat Forman za jej zrozumienie dla pośpiechu, z jakim trzeba było przygotowywać książkę i za wprowadzone poprawki oraz Robertowi L. Glassowi za wskazanie odpowiednich źródeł informacji.

Chcę także podziękować kilku osobom, z którymi odbyłem ważne rozmowy, gdy prace nad tą książką znajdowały się we wczesnej fazie i które także przeglądały wersję roboczą książki. Dziękuję Wayne'owi Beardsleyowi za pomoc w nadawaniu tonu książce i kierowaniu jej treścią, Brianowi Daugherty'emu za wskazanie powielonych informacji i sprawienie, że ta książka nie liczy o 500 stron więcej niż obecnie, Hankowi Meuretowi za zwrócenie szczególnej uwagi na przykłady kodu źródłowego oraz szczegóły kodowania, Gregowi Hitchcockowi za wskazanie miejsc, w których bezkrytycznie zawierzyłem stereotypom tworzenia oprogramowania oraz Alowi Corwinowi za to, że tak żywiołowo bronił lepszych metod tworzenia oprogramowania.

Szczególne podziękowania składam Tony'emu Garalandowi za to, że niezwykle uważnie przejrzał tę książkę. Jego uwagi znalazły odzwierciedlenie niemal na każdej stronie. Przypominał mi także, iż metody konstruowania kodu zasługują na poświęcenie im większej uwagi, niż to ma zazwyczaj miejsce. Jego komentarze — na przykład te, o innych osobach przeglądających wersję roboczą — stanowią idealny przykład konstruktywnego krytycyzmu.

Wszystkie osoby przeglądające tę książkę zakwestionowały co najmniej kilka podanych w niej zaleceń. Ponieważ uważam, że książka taka jak ta powinna zawierać wyraźne piętno jednego autora, a nie mgliste oznaki pracy wielu osób, to do mnie należała ostateczna decyzja odnośnie sposobu rozwiązania każdej kwestii. W związku z tym, mimo znacznego wkładu pracy innych osób, to ja ponoszę odpowiedzialność za wszelkie błędy, które znalazły się w treści lub które dotyczą zagadnień technicznych. Wszelkie komentarze można mi zgłaszać, pisząc na adres wydawnictwa Microsoft Press lub wysyłając wiadomość na adres [steve.mcconnell@construx.com](mailto:steve.mcconnell@construx.com).

*Tacoma, Washington  
Nowy Rok, 1993*

## **O Autorze**

Steve McConnell jest prezesem i głównym informatykiem firmy Construx Software. Swoją czas dzieli między kierowanie projektami programistycznymi, prowadzenie szkoleń oraz pisanie książek i artykułów. Jest także redaktorem naczelnym czasopisma *IEEE Software*. Oprócz książki „Programista doskonały” napisał także następujące pozycje: „Rapid Development”, „Software Project Survival Guide” oraz właśnie wznowioną „After the Gold Rush”.