

## **Wstęp**

- Do kogo jest kierowana ta książka
  - Doświadczeni programiści
  - Programiści-samoucy
  - Studenci
- Główne korzyści płynące z lektury tej książki
- Inne miejsca, w których można znaleźć te informacje
- Powód napisania tej książki
  - Lekceważenie zagadnienia konstruowania
  - Duże znaczenie konstruowania
  - Nie ma innej porównywalnej książki
- Podziękowania
- O Autorze

## **Część I**

### **Rozdział 1 Konstruowanie oprogramowania**

- 1.1 Na czym polega konstruowanie oprogramowania
- 1.2 Znaczenie konstruowania oprogramowania
- 1.3 Podsumowanie
  - 1.3.1 Warto zapamiętać

### **Rozdział 2 Metafory ułatwiające zrozumienie znaczenia programowania**

- 2.1 Siła metafor
- 2.2 Sposób używania metafor dotyczących oprogramowania
- 2.3 Najczęściej spotykane metafory dotyczące oprogramowania
  - 2.3.1 Styl pisania programów – pisanie kodu programu
  - 2.3.2 Hodowanie oprogramowania – system wrastający
  - 2.3.3 Hodowanie pereł – system przyrastający
  - 2.3.4 Konstruowanie oprogramowania – budowanie programów
  - 2.3.5 Używanie metod programowania – bank pomysłów
  - 2.3.6 Łączenie metafor
- 2.4 Podsumowanie
  - 2.4.1 Warto zapamiętać
  - 2.4.2 Warto przeczytać

### **Rozdział 3 Czynności poprzedzające konstruowanie programu**

- 3.1 Znaczenie czynności wstępnych
  - 3.1.1 Przykłady niepełnego przygotowania
  - 3.1.2 Całkowicie przekonujący argument przemawiający za przestrzeganiem wymagań wstępnych przed rozpoczęciem konstruowania
    - 3.1.2.1 Odwoływanie się do logiki
    - 3.1.2.2 Odwoływanie się do analogii
    - 3.1.2.3 Odwoływanie się do danych
    - 3.1.2.4 Test na gotowość szefa
- 3.2 Zasada definiowania problemu
- 3.3 Zasada określania wymagań
  - 3.3.1 Dlaczego warto precyzować formalnie wymagania
  - 3.3.2 Mit stabilnych wymagań
  - 3.3.3 Podejście do wymagań zmienia się w trakcie konstruowania
  - 3.3.4 Pytania kontrolne
    - 3.3.4.1 Wymagania
    - 3.3.4.2 Treść wymagań
    - 3.3.4.3 Kompletność wymagań
    - 3.3.4.4 Jakość wymagań
  - 3.3.5 Warto przeczytać
- 3.4 Zasada wybierania architektury
  - 3.4.1 Typowe komponenty architektury
    - 3.4.1.1 Układ programu

- 3.4.1.2 Zmiana strategii
- 3.4.1.3 Kupowanie a budowanie
- 3.4.1.4 Główne struktury danych
- 3.4.1.5 Główne algorytmy
- 3.4.1.6 Główne obiekty
- 3.4.1.7 Funkcje typowe
- 3.4.1.8 Obsługa błędów
- 3.4.1.9 Odporność
- 3.4.1.10 Wydajność
- 3.4.1.11 Całkowita jakość w architekturze
- 3.4.2 Pytania kontrolne
  - 3.4.2.1 Architektura
- 3.5 Zasada wybierania języka programowania
  - 3.5.1 Opis języków programowania
    - 3.5.1.1 Ada
    - 3.5.1.2 Asembler
    - 3.5.1.3 Basic
    - 3.5.1.4 C
    - 3.5.1.5 C++
    - 3.5.1.6 Fortran
    - 3.5.1.7 Pascal
  - 3.5.2 Krótka instrukcja wybierania języka programowania
- 3.6 Konwencje w programowaniu
- 3.7 Czas poświęcany na wykonywanie czynności wstępnych
- 3.8 Adaptowanie zasad do określonego projektu
- 3.9 Podsumowanie
  - 3.9.1 Warto zapamiętać

## Część II

### Rozdział 4 Czynności wykonywane podczas budowania podprogramów

- 4.1 Spis czynności wykonywanych podczas budowania podprogramów
- 4.2 Zapis PDL dla profesjonalistów
- 4.3 Projektowanie podprogramów
- 4.4 Tworzenie kodu podprogramów
- 4.5 Formalne sprawdzanie kodu
- 4.6 Podsumowanie
  - 4.6.1 Pytania kontrolne
    - 4.6.1.1 Konstruowanie podprogramów
  - 4.6.2 Warto zapamiętać

### Rozdział 5 Cechy charakterystyczne dobrych podprogramów

- 5.1 Powody tworzenia podprogramów
  - 5.1.1 Operacje, które wydają się za mało skomplikowane, aby je umieszczać w podprogramie
  - 5.1.2 Podsumowanie powodów tworzenia podprogramów
- 5.2 Dobre nazwy podprogramów
- 5.3 Duża spójność
  - 5.3.1 Spójność dopuszczalna
  - 5.3.2 Spójność niedopuszczalna
  - 5.3.3 Przykłady spójności
- 5.4 Luźne powiązania
  - 5.4.1 Kryteria określania powiązań
  - 5.4.2 Poziome powiązania
  - 5.4.3 Przykłady powiązań
- 5.5 Jak długi powinien być podprogram
- 5.6 Programowanie defensywne
  - 5.6.1 Używanie asercji
  - 5.6.2 Złe dane wejściowe nie muszą prowadzić do powstania złych danych wyjściowych
  - 5.6.3 Obsługiwanie sytuacji wyjątkowych

- 5.6.4 Przewidywanie zmian
- 5.6.5 Plan usuwania mechanizmów używanych podczas poprawiania błędów
- 5.6.6 Wczesne wprowadzanie narzędzi ułatwiających wykrywanie błędów
- 5.6.7 Izolowanie obszarów ze skutkami spowodowanymi przez błędy
- 5.6.8 Sprawdzanie wartości zwracanych przez funkcje
- 5.6.9 Ustalanie zakresu działania mechanizmów programowania defensywnego pozostawionych w kodzie ostatecznej wersji programu
- 5.6.10 Zachowanie ostrożności przy programowaniu defensywnym
- 5.7 Sposób używania parametrów podprogramów
- 5.8 Używanie funkcji
  - 5.8.1 Okoliczności używania funkcji i procedur
  - 5.8.2 Ryzyko związane z używaniem funkcji
- 5.9 Makropolecenia
- 5.10 Podsumowanie
  - 5.10.1 Pytania kontrolne
    - 5.10.1.1 Dobre podprogramy
  - 5.10.2 Warto zapamiętać

## **Rozdział 6 Trzy czwarte programistów używa modułów**

- 6.1 Modułowość – spójność i powiązania
  - 6.1.1 Spójność modułu
  - 6.1.2 Powiązania między modułami
- 6.2 Ukrywanie danych
  - 6.2.1 Sekrety i prawo do prywatności
  - 6.2.2 Przykład ukrywania danych
  - 6.2.3 Najczęściej występujące rodzaje sekretów
    - 6.2.3.1 Obszary, w których trzeba wprowadzić zmiany
    - 6.2.3.2 Skomplikowane dane
    - 6.2.3.3 Skomplikowany sposób działania
    - 6.2.3.4 Operacje z poziomu języka programowania
  - 6.2.4 Przeszkody występujące podczas ukrywania danych
    - 6.2.4.1 Nadmierne rozpowszechnianie danych
    - 6.2.4.2 Zapętlenia
    - 6.2.4.3 Dane modułu mylone z danymi globalnymi
    - 6.2.4.4 Dostrzegane ograniczenia wydajności
  - 6.2.5 Warto przeczytać
- 6.3 Powody, dla których warto tworzyć moduły
- 6.4 Implementowanie modułów w różnych językach programowania
  - 6.4.1 Dostępne w językach programowania mechanizmy do obsługi modułowości
  - 6.4.2 Podsumowanie mechanizmów dostępnych w językach programowania
    - 6.4.2.1 Mechanizmy dostępne w językach Ada i Modula-2
    - 6.4.2.2 Mechanizmy dostępne w językach obiektowych
    - 6.4.2.3 Mechanizmy dostępne w Pascalu
    - 6.4.2.4 Mechanizmy dostępne w C
    - 6.4.2.5 Mechanizmy dostępne w Fortranie
  - 6.4.3 Udawanie modułowości
    - 6.4.3.1 Umieszczanie danych i podprogramów w modułach
    - 6.4.3.2 Zapewnianie prywatności wewnętrznych podprogramów modułu
    - 6.4.3.3 Zapewnienie prywatności wewnętrznych danych modułu
- 6.5 Podsumowanie
  - 6.5.1 Pytania kontrolne
    - 6.5.1.1 Jakość modułu
  - 6.5.2 Warto zapamiętać

## **Rozdział 7 Projektowanie wysokiego poziomu przy konstruowaniu programów**

- 7.1 Wstęp do projektowania oprogramowania
  - 7.1.1 Projektowanie małych i dużych programów
  - 7.1.2 Poziomy projektu
    - 7.1.2.1 Poziom pierwszy – podział na podsystemy

- 7.1.2.2 Poziom drugi – podział na moduły
- 7.1.2.3 Poziom trzeci – podział na podprogramy
- 7.1.2.4 Poziom czwarty – projekt poszczególnych podprogramów
- 7.1.3 Projektowanie na etapie konstruowania
  - 7.1.3.1 Projekt poszczególnych podprogramów
  - 7.1.3.2 Podział na podprogramy
  - 7.1.3.3 Podział na moduły
  - 7.1.3.4 Podział na podsystemy
- 7.2 Projekt strukturalny
  - 7.2.1 Wybieranie komponentów, które zostaną umieszczone w modułach
    - 7.2.1.1 Dekompozycja zstępująca
    - 7.2.1.2 Składanie wstępujące
    - 7.2.1.3 Zestawienie obydwóch metod
  - 7.2.2 Warto przeczytać
- 7.3 Projekt obiektowy
  - 7.3.1 Główne założenia
    - 7.3.1.1 Abstrakcyjność
    - 7.3.1.2 Kapsułkowanie
    - 7.3.1.3 Modułowość
    - 7.3.1.4 Hierarchia i dziedziczenie
    - 7.3.1.5 Obiekty i klasy
  - 7.3.2 Czynności wykonywane podczas projektowania obiektowego
  - 7.3.3 Elementy typowego projektu obiektowego
  - 7.3.4 Warto przeczytać
- 7.4 Omówienie najpopularniejszych metod pracy
  - 7.4.1 Kiedy należy używać projektowania strukturalnego
  - 7.4.2 Kiedy należy używać ukrywania danych
  - 7.4.3 Kiedy należy używać projektowania obiektowego
- 7.5 Projektowanie iteracyjne
  - 7.5.1 Na czym polega iterowanie
  - 7.5.2 Projektowanie jest procesem nieuporządkowanym
  - 7.5.3 Projektowanie jest procesem złośliwym
  - 7.5.4 Projektowanie jest procesem heurystycznym
  - 7.5.5 Cechy charakterystyczne projektu
- 7.6 Podsumowanie
  - 7.6.1 Pytania kontrolne
    - 7.6.1.1 Projektowanie wysokiego poziomu
  - 7.6.2 Warto przeczytać
    - 7.6.2.1 Projektowanie oprogramowania
    - 7.6.2.2 Alternatywne metody projektowania oprogramowania
    - 7.6.2.3 Projektowanie w kategoriach ogólnych
  - 7.6.3 Warto zapamiętać

## **Część III**

### **Rozdział 8 Tworzenie danych**

- 8.1 Sposoby zapisu danych
  - 8.1.1 Test znajomości nazw struktur danych
- 8.2 Powody tworzenia własnych typów danych
- 8.3 Wskazówki dotyczące tworzenia własnych typów danych
- 8.4 Łatwe deklarowanie zmiennych
  - 8.4.1 Używanie szablonów do deklarowania zmiennych
  - 8.4.2 Deklaracje domyślne
- 8.5 Wskazówki dotyczące inicjalizacji danych
- 8.6 Podsumowanie
  - 8.6.1 Pytania kontrolne
  - 8.6.2 Warto zapamiętać

## **Rozdział 9 Znaczenie nazw danych**

- 9.1 Rozważania dotyczące właściwego doboru nazw
  - 9.1.1 Najważniejsza zasada dobierania nazw
  - 9.1.2 Sformułowanie problemu
  - 9.1.3 Optymalna długość nazwy
  - 9.1.4 Znaczenie zasięgu zmiennych nazw
  - 9.1.5 Określenia komputerowe umieszczane w nazwach
  - 9.1.6 Typowe przeciwstawności w nazwach zmiennych
- 9.2 Nadawanie nazw określonym typom danych
  - 9.2.1 Nadawanie nazw indeksom pętli
  - 9.2.2 Nadawanie nazw zmiennym stanu
  - 9.2.3 Nadawanie nazw zmiennym tymczasowym
  - 9.2.4 Nadawanie nazw zmiennym logicznym
  - 9.2.5 Nadawanie nazw typom wyliczeniowym
  - 9.2.6 Nadawanie nazw wartościom stałym
- 9.3 Siła konwencji nazewniczych
  - 9.3.1 Czemu służą konwencje
  - 9.3.2 Kiedy należy opracować konwencję nazewniczą
  - 9.3.3 Stopień sformalizowania
- 9.4 Nieformalne konwencje nazewnicze
  - 9.4.1 Wskazówki dotyczące tworzenia konwencji niezależnych od języka programowania
  - 9.4.2 Wskazówki dotyczące konwencji narzucanych przez język programowania
    - 9.4.2.1 Konwencje obowiązujące w języku C
    - 9.4.2.2 Konwencje obowiązujące w Pascalu
    - 9.4.2.3 Konwencje obowiązujące w Fortranie
  - 9.4.3 Przykładowe konwencje nazewnicze
- 9.5 Węgierska konwencja nazewnicza
  - 9.5.1 Typy podstawowe
  - 9.5.2 Prefiksy
  - 9.5.3 Oznaczenia klasyfikujące
  - 9.5.4 Przykłady nazw zgodnych z węgierską konwencją nazewniczą
  - 9.5.5 Zalety płynące z używania węgierskiej konwencji nazewniczej
  - 9.5.6 Wady węgierskiej konwencji nazewniczej
- 9.6 Tworzenie krótkich i czytelnych nazw
  - 9.6.1 Ogólne wskazówki dotyczące skracania nazw
  - 9.6.2 Skróty fonetyczne
  - 9.6.3 Reguły rządzące skrótami
- 9.7 Rodzaje nazw, których należy unikać
- 9.8 Podsumowanie
  - 9.8.1 Pytania kontrolne
    - 9.8.1.1 Nadawanie nazw danym
  - 9.8.2 Warto zapamiętać

## **Rozdział 10 Ogólne zagadnienia dotyczące używania zmiennych**

- 10.1 Zasięg
- 10.2 Trwałość
- 10.3 Moment powiązania
- 10.4 Związek między strukturami danych i konstrukcjami sterującymi
- 10.5 Używanie jednej zmiennej tylko w jednym celu
- 10.6 Zmienne globalne
  - 10.6.1 Typowe problemy związane z danymi globalnymi
  - 10.6.2 Powody używania zmiennych globalnych
  - 10.6.3 Sposób zmniejszenia ryzyka związanego z używaniem danych globalnych
  - 10.6.4 Używanie procedur dostępu zamiast danych globalnych
    - 10.6.4.1 Zalety procedur dostępu
    - 10.6.4.2 Sposób używania procedur dostępu
- 10.7 Podsumowanie
  - 10.7.1 Pytania kontrolne
    - 10.7.1.1 Ogólne rozważania na temat używania danych

- 10.7.1.2 Dane w znaczeniu ogólnym
- 10.7.1.3 Dane globalne
- 10.7.2 Warto zapamiętać

## **Rozdział 11 Podstawowe typy danych**

- 11.1 Liczby
- 11.2 Liczby całkowite
- 11.3 Liczby zmiennoprzecinkowe
- 11.4 Znaki i napisy
  - 11.4.1 Napisy w języku C
- 11.5 Zmienne logiczne
- 11.6 Typy wyliczeniowe
  - 11.6.1 Sytuacja, gdy język programowania nie pozwala na używanie typów wyliczeniowych
- 11.7 Nazwane wartości stałe
- 11.8 Tablice
- 11.9 Wskaźniki
  - 11.9.1 Co to są wskaźniki
    - 11.9.1.1 Umieszczenie w pamięci
    - 11.9.1.2 Sposób interpretowania zawartości pamięci
  - 11.9.2 Ogólne wskazówki na temat wskaźników
  - 11.9.3 Wskaźniki w języku C
- 11.10 Podsumowanie
  - 11.10.1 Pytania kontrolne
    - 11.10.1.1 Podstawowe typy danych
  - 11.10.2 Warto zapamiętać

## **Rozdział 12 Złożone typy danych**

- 12.1 Rekordy i struktury
- 12.2 Mechanizmy tablicowe
  - 12.2.1 Ogólne zagadnienia dotyczące mechanizmów tablicowych
  - 12.2.2 Dostęp bezpośredni
    - 12.2.2.1 Przykład z dniami miesiąca
    - 12.2.2.2 Przykład ze stawkami ubezpieczenia
    - 12.2.2.3 Przykład elastycznego formatu komunikatów
    - 12.2.2.4 Przygotowywanie kluczy wyszukiwania
  - 12.2.3 Dostęp przy użyciu indeksów
  - 12.2.4 Dostęp schodkowy
  - 12.2.5 Inne przykłady metod wybierania elementów z tablicy
- 12.3 Abstrakcyjne typy danych
  - 12.3.1 Przykład ilustrujący konieczność użycia abstrakcyjnego typu danych
  - 12.3.2 Korzyści płynące z używania abstrakcyjnych typów danych
  - 12.3.3 Kolejne przykłady abstrakcyjnych typów danych
  - 12.3.4 Obsługiwanie wielu instancji danych przy użyciu abstrakcyjnych typów danych
  - 12.3.5 Łączenie ze sobą poziomów abstrakcji (nie zalecane)
    - 12.3.5.1 Rekordy otwarte i zamknięte
  - 12.3.6 Abstrakcyjne typy danych a ukrywanie danych, modułów i obiektów
  - 12.3.7 Funkcje do obsługi abstrakcyjnych typów danych udostępnione w językach programowania
- 12.4 Podsumowanie
  - 12.4.1 Warto zapamiętać

## **Część IV**

### **Rozdział 13 Tworzenie czytelnego kodu**

- 13.1 Wyrażenia, które muszą występować w określonej kolejności
- 13.2 Wyrażenia, których kolejność nie ma znaczenia
  - 13.2.1 Nadawanie kodowi czytelnej postaci
  - 13.2.2 Umieszczanie w jednym miejscu odwołań do zmiennych
  - 13.2.3 Maksymalne skracanie czasu życia zmiennych
    - 13.2.3.1 Rekordy otwarte i zamknięte
  - 13.2.4 Grupowanie powiązanych wyrażeń

- 13.3 Podsumowanie
  - 13.3.1 Pytania kontrolne
    - 13.3.1.1 Konstruowanie czytelnego kodu
  - 13.3.2 Warto zapamiętać

## Rozdział 14 Używanie instrukcji warunkowych

- 14.1 Instrukcje **if**
  - 14.1.1 Zwykłe instrukcje **if-then**
  - 14.1.2 Konstrukcje **if-then-else**
- 14.2 Instrukcje **case**
  - 14.2.1 Wybór optymalnej kolejności przypadków
  - 14.2.2 Porady dotyczące używania instrukcji **case**
- 14.3 Podsumowanie
  - 14.3.1 Pytania kontrolne
    - 14.3.1.1 Instrukcje warunkowe
  - 14.3.2 Warto zapamiętać

## Rozdział 15 Pętle sterujące

- 15.1 Wybór rodzaju pętli
  - 15.1.1 Okoliczności użycia pętli **while**
    - 15.1.1.1 Pętle z testem wykonywanym na początku
    - 15.1.1.2 Pętle z testem wykonywanym na końcu
  - 15.1.2 Okoliczności użycia pętli z instrukcją wyjścia
    - 15.1.2.1 Typowe pętle z instrukcją wyjścia
    - 15.1.2.2 Nietypowe pętle z instrukcją wyjścia
  - 15.1.3 Okoliczności użycia pętli **for**
- 15.2 Sterowanie działaniem pętli
  - 15.2.1 Rozpoczynanie wykonywania pętli
  - 15.2.2 Wykonywanie operacji wewnątrz pętli
  - 15.2.3 Zakończenie działania pętli
    - 15.2.3.1 Używanie instrukcji break i continue
  - 15.2.4 Sprawdzanie punktów krańcowych
  - 15.2.5 Używanie zmiennych pętli
  - 15.2.6 Najbardziej odpowiednia długość pętli
- 15.3 Łatwe tworzenie pętli od środka
- 15.4 Powiązanie między pętlami i tablicami
- 15.5 Podsumowanie
  - 15.5.1 Pytania kontrolne
    - 15.5.1.1 Pętle
  - 15.5.2 Warto zapamiętać

## Rozdział 16 Nietypowe konstrukcje sterujące

- 16.1 Instrukcja **goto**
  - 16.1.1 Argumenty przeciwko używaniu instrukcji **goto**
  - 16.1.2 Argumenty przemawiające za używaniem instrukcji **goto**
  - 16.1.3 Sztuczna debata nad instrukcją **goto**
  - 16.1.4 Obsługiwanie błędów a instrukcje **goto**
    - 16.1.4.1 Porównanie metod
  - 16.1.5 Instrukcje **goto** a współużytkowanie kodu w blokach **else**
  - 16.1.6 Podsumowanie wskazówek dotyczących używania instrukcji **goto**
  - 16.1.7 Warto przeczytać
- 16.2 Instrukcja **return**
- 16.3 Rekurencja
  - 16.3.1 Przykład rekurencji
  - 16.3.2 Porady dotyczące stosowania rekurencji
- 16.4 Podsumowanie
  - 16.4.1 Pytania kontrolne
    - 16.4.1.1 Nietypowe konstrukcje sterujące
  - 16.4.2 Warto zapamiętać

## Rozdział 17 Ogólne zagadnienia dotyczące sterowania

- 17.1 Wyrażenia logiczne
  - 17.1.1 Używanie wartości **True** lub **False** w testach logicznych
  - 17.1.2 Upraszczenie skomplikowanych wyrażeń
  - 17.1.3 Formułowanie wyrażeń logicznych o sensie pozytywnym
  - 17.1.4 Używanie nawiasów do upraszczania wyrażeń logicznych
  - 17.1.5 Sposób obliczania wyrażeń logicznych
  - 17.1.6 Zapisywanie wyrażeń liczbowych w kolejności rosnącej
  - 17.1.7 Wskazówki dotyczące porównywania wartości z zerem w języku C
  - 17.1.8 Najczęstsze problemy związane z wyrażeniami logicznymi
- 17.2 Wyrażenia złożone (bloki)
- 17.3 Instrukcja pusta
- 17.4 Zapobieganie niebezpiecznie głębokiemu zagnieżdżaniu
- 17.5 Zalety programowania strukturalnego
  - 17.5.1 Korzyści płynące z programowania strukturalnego
  - 17.5.2 Trzy elementy programowania strukturalnego
    - 17.5.2.1 Sekwencja
    - 17.5.2.2 Mechanizm wybierania elementów
    - 17.5.2.3 Iteracja
- 17.6 Emulowanie konstrukcji strukturalnych za pomocą instrukcji **goto**
  - 17.6.1 Emulowanie konstrukcji **if-then-else**
  - 17.6.2 Emulowanie instrukcji **case**
  - 17.6.3 Emulowanie pętli **while**
  - 17.6.4 Ogólne zagadnienia dotyczące emulowania konstrukcji sterujących
- 17.7 Konstrukcje sterujące a złożoność
  - 17.7.1 Jak istotna jest złożoność
  - 17.7.2 Ogólne wskazówki dotyczące ograniczania stopnia skomplikowania
    - 17.7.2.1 Sposób określania stopnia złożoności
    - 17.7.2.2 Interpretowanie wyników pomiaru złożoności
  - 17.7.3 Inne kryteria złożoności
- 17.8 Podsumowanie
  - 17.8.1 Pytania kontrolne
    - 17.8.1.1 Zagadnienia dotyczące konstrukcji sterujących
  - 17.8.2 Warto zapamiętać

## Część V

### Rozdział 18 Układ i styl

- 18.1 Podstawowe informacje o układzie kodu źródłowego
  - 18.1.1 Przykłady błędnego układu kodu źródłowego
  - 18.1.2 Teoria formatowania kodu źródłowego
  - 18.1.3 Analizowanie kodu programu przez człowieka i przez komputer
  - 18.1.4 Jaką wartość ma dobry układ
  - 18.1.5 Ortodoksyjne podejście do układu kodu
  - 18.1.6 Rola dobrego układu
    - 18.1.6.1 Jak korzystać z przedstawionej listy kryteriów
- 18.2 Techniki używane podczas przygotowywania układu
  - 18.2.1 Odstęp
  - 18.2.2 Nawiasy
- 18.3 Style układu
  - 18.3.1 Bloki podstawowe
  - 18.3.2 Układ z głębokimi wcięciami
  - 18.3.3 Zastępowanie bloków podstawowych
  - 18.3.4 Słowa kluczowe **begin** i **end** jako granice bloku
  - 18.3.5 Jaki styl jest najlepszy
- 18.4 Układ instrukcji sterujących
  - 18.4.1 Formatowanie bloków instrukcji sterujących
  - 18.4.2 Pozostałe zagadnienia
- 18.5 Układ poszczególnych wyrażeń

- 18.5.1 Długość wyrażeń
- 18.5.2 Stosowanie spacji w celu poprawienia czytelności
- 18.5.3 Wyrównywanie grup wzajemnie powiązanych przypisań
- 18.5.4 Formatowanie podzielonych poleceń
- 18.5.5 Umieszczanie tylko jednego polecenia w wierszu
- 18.5.6 Formatowanie deklaracji zmiennych
- 18.6 Układ komentarzy
- 18.7 Układ podprogramów
- 18.8 Układ plików, modułów i programów
- 18.9 Podsumowanie
  - 18.9.1 Pytania kontrolne
    - 18.9.1.1 Układ
  - 18.9.2 Warto przeczytać
  - 18.9.3 Warto zapamiętać

## **Rozdział 19 Kod niewymagający komentarzy**

- 19.1 Dokumentacja programu niedołączona do jego kodu
- 19.2 Styl programowania zastępujący dokumentację
  - 19.2.1 Pytania kontrolne
    - 19.2.1.1 Kod niewymagający komentarzy
- 19.3 Komentować albo nie komentować
- 19.4 Efektywne komentowanie kodu
  - 19.4.1 Rodzaje komentarzy
  - 19.4.2 Efektywne stosowanie komentarzy
  - 19.4.3 Optymalna liczba komentarzy w kodzie programu
- 19.5 Sposoby stosowania komentarzy
  - 19.5.1 Komentowanie pojedynczych wierszy
    - 19.5.1.1 Kod niewymagający komentarzy
  - 19.5.2 Komentowanie akapitów w kodzie źródłowym
  - 19.5.3 Komentowanie deklaracji zmiennych
  - 19.5.4 Komentowanie instrukcji sterujących
  - 19.5.5 Komentowanie podprogramów
  - 19.5.6 Komentowanie plików, modułów i programów
    - 19.5.6.1 Ogólne wskazówki dotyczące dokumentowania plików
    - 19.5.6.2 Komentarze w kodzie programu a układ książki
- 19.6 Podsumowanie
  - 19.6.1 Pytania kontrolne
    - 19.6.1.1 Techniki tworzenia dobrych komentarzy
  - 19.6.2 Warto przeczytać
  - 19.6.3 Warto zapamiętać

## **Rozdział 20 Narzędzia programistyczne**

- 20.1 Narzędzia projektowe
- 20.2 Narzędzia ułatwiające pracę z kodem źródłowym
  - 20.2.1 Edycja
    - 20.2.1.1 Edytowanie kodu
    - 20.2.1.2 Narzędzia umożliwiające zamienianie napisów w wielu plikach
    - 20.2.1.3 Narzędzia służące do porównywania plików
    - 20.2.1.4 Narzędzia poprawiające wygląd kodu źródłowego
    - 20.2.1.5 Szablony
  - 20.2.2 Przeglądanie
    - 20.2.2.1 Narzędzia służące do przeglądania kodu
    - 20.2.2.2 Narzędzia służące do wyszukiwania napisów w wielu plikach
    - 20.2.2.3 Narzędzia służące do tworzenia odnośników
    - 20.2.2.4 Narzędzia służące do tworzenia struktury wywołań
  - 20.2.3 Badanie jakości kodu
    - 20.2.3.1 Narzędzia sprawdzające składnię i poprawność logiczną kodu
    - 20.2.3.2 Narzędzia pomiaru jakości kodu
  - 20.2.4 Modyfikowanie struktury kodu źródłowego

- 20.2.4.1 Narzędzia służące do modyfikowania struktury programu
- 20.2.4.2 Narzędzia służące do tłumaczenia kodu
- 20.2.5 Sprawdzanie wersji kodu
- 20.2.6 Słowniki danych
- 20.3 Narzędzia ułatwiające pracę z plikami wykonywalnymi
  - 20.3.1 Tworzenie kodu
    - 20.3.1.1 Konsolidatory
    - 20.3.1.2 Biblioteki kodu
    - 20.3.1.3 Narzędzia przeznaczone do tworzenia oprogramowania
    - 20.3.1.4 Narzędzia służące do obsługi makropoleczeń
  - 20.3.2 Usuwanie błędów
  - 20.3.3 Testowanie
  - 20.3.4 Dopracowywanie kodu
    - 20.3.4.1 Narzędzia testujące wydajność
    - 20.3.4.2 Kod w asemblerze i dezasemblerze
- 20.4 Środowiska narzędziowe
  - 20.4.1 Unix
  - 20.4.2 CASE
  - 20.4.3 APSE
- 20.5 Tworzenie własnych narzędzi programistycznych
  - 20.5.1.1 Narzędzia dostosowane do konkretnego projektu
  - 20.5.1.2 Skrypty
  - 20.5.1.3 Narzędzia rozszerzające możliwości plików wsadowych
- 20.6 Idealne środowisko programistyczne
  - 20.6.1.1 Integracja
  - 20.6.1.2 Obsługiwane języki programowania
  - 20.6.1.3 Szczegółowe odnośniki
  - 20.6.1.4 Interakcyjne widoki struktury programu
  - 20.6.1.5 Interakcyjne formatowanie
  - 20.6.1.6 Dokumentacja
  - 20.6.1.7 Zwiększanie wydajności
  - 20.6.1.8 Stan parametrów środowiska
- 20.6.1 Podstawowe zalety środowiska Szewc
- 20.7 Podsumowanie
  - 20.7.1 Warto przeczytać
  - 20.7.2 Warto zapamiętać

## **Rozdział 21 Wpływ wielkości programu na proces jego tworzenia**

- 21.1 Wielkość projektów
- 21.2 Wpływ wielkości projektu na proces jego tworzenia
  - 21.2.1 Komunikowanie się a wielkość projektu
  - 21.2.2 Rodzaj wykonywanych działań a wielkość projektów
  - 21.2.3 Metodologia a wielkość projektów
  - 21.2.4 Programy, produkty, systemy i produkty systemowe
- 21.3 Wpływ wielkości projektu na powstawanie błędów
- 21.4 Wpływ wielkości projektu na wydajność pracy nad nim
- 21.5 Podsumowanie
  - 21.5.1 Warto przeczytać
  - 21.5.2 Warto zapamiętać

## **Rozdział 22 Zarządzanie procesem tworzenia oprogramowania**

- 22.1 Mechanizmy zachęcające do tworzenia dobrego kodu źródłowego
  - 22.1.1 Zagadnienia związane z wprowadzaniem standardów
  - 22.1.2 Techniki
- 22.2 Zarządzanie konfiguracją
  - 22.2.1 Czym jest zarządzanie konfiguracją
  - 22.2.2 Zmiany wprowadzane podczas projektowania oprogramowania
  - 22.2.3 Zmiany wprowadzane podczas tworzenia kodu źródłowego
  - 22.2.4 Plan wykonywania kopii zapasowych

- 22.2.5 Pytania kontrolne
  - 22.2.5.1 Zarządzanie konfiguracją
- 22.2.6 Warto przeczytać
- 22.3 Przygotowywanie planu tworzenia oprogramowania
  - 22.3.1 Metody oszacowywania
  - 22.3.2 Oszacowywanie czasu potrzebnego na przygotowanie programu
  - 22.3.3 Wpływ różnych czynników na harmonogram prac nad projektem
  - 22.3.4 Oszacowania a sprawowanie kontroli nad projektem
  - 22.3.5 Co robić, gdy prace nad projektem się opóźniają
  - 22.3.6 Warto przeczytać
- 22.4 Miary
  - 22.4.1 Warto przeczytać
- 22.5 Programista też człowiek
  - 22.5.1 Sposoby spędzania czasu przez programistów
  - 22.5.2 Zmienność wydajności pracy i jej jakości
    - 22.5.2.1 Różnice między wydajnością pracy i jej jakością u różnych programistów
    - 22.5.2.2 Różnice między wydajnością pracy i jej jakością w różnych zespołach
  - 22.5.3 Zagadnienia kontrowersyjne
  - 22.5.4 Miejsce pracy
  - 22.5.5 Warto przeczytać
- 22.6 Współpraca ze swoim szefem
- 22.7 Podsumowanie
  - 22.7.1 Warto przeczytać
  - 22.7.2 Warto zapamiętać

## **Część VI**

### **Rozdział 23 Jakość oprogramowania**

- 23.1 Charakterystyka jakości oprogramowania
- 23.2 Metody podnoszenia jakości oprogramowania
  - 23.2.1 Wskazywanie celów
- 23.3 Względna wydajność poszczególnych metod
  - 23.3.1 Liczba wykrytych błędów
  - 23.3.2 Koszt znajdowania błędów
  - 23.3.3 Koszt usuwania błędów
- 23.4 Kiedy należy stosować metody kontroli jakości
- 23.5 Podstawowa zasada związana z jakością oprogramowania
- 23.6 Podsumowanie
  - 23.6.1 Pytania kontrolne
    - 23.6.1.1 Program działań podejmowanych w celu podniesienia jakości
  - 23.6.2 Warto przeczytać
  - 23.6.3 Warto zapamiętać

### **Rozdział 24 Prowadzenie przeglądów**

- 24.1 Rola przeglądu w procesie podnoszenia jakości oprogramowania
  - 24.1.1 Przeglądy uzupełniają inne metody podnoszenia jakości
  - 24.1.2 Przeglądy umożliwiają propagowanie zasad obowiązujących w przedsiębiorstwie i wzajemne przekazywanie doświadczeń
  - 24.1.3 Przeglądy umożliwiające ocenianie jakości i postępów prac
  - 24.1.4 Przeglądy ułatwiają zarówno pracę prowadzoną przed etapem tworzenia kodu, jak i działania podejmowane później
- 24.2 Badania formalne
  - 24.2.1 Wyniki przeprowadzonych badań
  - 24.2.2 Zadania podczas badania formalnego
  - 24.2.3 Ogólna procedura przeprowadzania badania formalnego
  - 24.2.4 Rola ambicji w badaniach formalnych
  - 24.2.5 Pytania kontrolne
    - 24.2.5.1 Efektywne badanie formalne
  - 24.2.6 Podsumowanie wiadomości na temat badania formalnego

- 24.3 Inne rodzaje przeglądów
  - 24.3.1 Badanie nieformalne
    - 24.3.1.1 Wyniki badania nieformalnego
    - 24.3.1.2 Porównanie badań formalnych z badaniami nieformalnymi
  - 24.3.2 Czytanie kodu
  - 24.3.3 Pokazy
- 24.4 Podsumowanie
  - 24.4.1 Warto przeczytać
    - 24.4.1.1 Badania formalne
    - 24.4.1.2 Badania nieformalne
    - 24.4.1.3 Przeglądy
  - 24.4.2 Warto zapamiętać

## **Rozdział 25 Testowanie poszczególnych fragmentów systemu**

- 25.1 Rola testowania poszczególnych fragmentów systemu w procesie podnoszenia jakości oprogramowania
  - 25.1.1 Testowanie podczas pracy nad kodem
- 25.2 Testowanie poszczególnych fragmentów systemu
- 25.3 Sposoby przeprowadzania testów
  - 25.3.1 Niepełne testy
  - 25.3.2 Podstawowe testowanie strukturalne
  - 25.3.3 Testowanie przepływu danych
    - 25.3.3.1 Kombinacje stanów danych
  - 25.3.4 Równoważne zestawy danych
  - 25.3.5 Zgadywanie błędów
  - 25.3.6 Analizowanie ograniczeń
    - 25.3.6.1 Złożone ograniczenia
  - 25.3.7 Klasy błędnych danych
  - 25.3.8 Klasy poprawnych danych
  - 25.3.9 Pytania kontrolne
    - 25.3.9.1 Zestawy danych testowych
  - 25.3.10 Korzystanie z zestawów danych, które ułatwiają samodzielne sprawdzanie wyników testów
- 25.4 Typowe błędy
  - 25.4.1 Podprogramy zawierające największą liczbę błędów
  - 25.4.2 Klasyfikacja błędów
  - 25.4.3 Liczba błędów spowodowanych podczas tworzenia kodu źródłowego
  - 25.4.4 Ilu błędów należy się spodziewać
  - 25.4.5 Błędy powstające w procesie testowania
- 25.5 Narzędzia ułatwiające testowanie
  - 25.5.1 Tworzenie zastępczych elementów ułatwiających testowanie poszczególnych podprogramów
  - 25.5.2 Narzędzia pozwalające na porównywanie wyników
  - 25.5.3 Narzędzia pozwalające na tworzenie danych testowych
  - 25.5.4 Narzędzia sprawdzające stopień przetestowania kodu programu
  - 25.5.5 Symboliczne narzędzia służące do usuwania błędów
  - 25.5.6 Narzędzia pozwalające na zakłócanie pracy systemu
  - 25.5.7 Bazy danych zawierające informacje na temat błędów
- 25.6 Usprawnianie przebiegu testów
  - 25.6.1 Planowanie testu
  - 25.6.2 Testowanie wtórne
- 25.7 Gromadzenie danych na temat przeprowadzonych testów
- 25.8 Podsumowanie
  - 25.8.1 Warto przeczytać
  - 25.8.2 Warto zapamiętać

## **Rozdział 26 Usuwanie błędów**

- 26.1 Zagadnienia związane z usuwaniem błędów
  - 26.1.1 Proces usuwania błędów a jakość oprogramowania
  - 26.1.2 Różnice w wydajności procesu usuwania błędów
  - 26.1.3 Traktowanie błędów jako nowych możliwości
  - 26.1.4 Nieskuteczne metody

- 26.1.4.1 Szatańskie porady
- 26.1.4.2 Metody usuwania błędów oparte na przesądach
- 26.2 Znajdowanie błędu
  - 26.2.1 Naukowa metoda usuwania błędów
    - 26.2.1.1 Wykrycie błędu za pomocą powtarzalnych testów
    - 26.2.1.2 Znalezienie przyczyny błędu
  - 26.2.2 Wskazówki dotyczące znajdowania błędów
  - 26.2.3 Błędy składniowe
- 26.3 Usuwanie błędu
- 26.4 Zagadnienia psychologiczne związane z usuwaniem błędów
  - 26.4.1 W jaki sposób nastawienie wpływa na dostrzegane błędy
  - 26.4.2 W jaki sposób dostrzegane różnice między elementami ułatwiają pracę
- 26.5 Narzędzia ułatwiające usuwanie błędów
  - 26.5.1 Narzędzia pozwalające na porównywanie kodu źródłowego
  - 26.5.2 Komunikaty o błędach wyświetlane przez kompilator
  - 26.5.3 Rozbudowane narzędzia służące do sprawdzania składni i poprawności logicznej
  - 26.5.4 Narzędzia testujące wydajność oprogramowania
  - 26.5.5 Elementy zastępcze
  - 26.5.6 Debugger
- 26.6 Podsumowanie
  - 26.6.1 Pytania kontrolne
    - 26.6.1.1 Usuwanie błędów
  - 26.6.2 Warto przeczytać
  - 26.6.3 Warto zapamiętać

## **Część VII**

### **Rozdział 27 Integrowanie systemu**

- 27.1 Znaczenie wybranej metody integrowania
- 27.2 Integrowanie systemu w jednym etapie a integrowanie przyrostowe
  - 27.2.1 Integrowanie w jednym etapie
  - 27.2.2 Integrowanie przyrostowe
    - 27.2.2.1 Ogólny opis
    - 27.2.2.2 Zalety integrowania przyrostowego
- 27.3 Metody przyrostowego integrowania systemu
  - 27.3.1 Integrowanie zstępujące
  - 27.3.2 Integrowanie wstępujące
  - 27.3.3 Integrowanie warstwowe
  - 27.3.4 Integrowanie z uwzględnieniem stopnia ryzyka
  - 27.3.5 Integrowanie podprogramów odpowiedzialnych za konkretną funkcję systemu
  - 27.3.6 Pytania kontrolne
    - 27.3.6.1 Integrowanie przyrostowe
- 27.4 Stopniowe tworzenie oprogramowania
  - 27.4.1 Analogie do przygotowań do wyprawy
  - 27.4.2 Ogólny opis
  - 27.4.3 Korzyści
  - 27.4.4 Podobieństwa do metody polegającej na opracowywaniu prototypów
  - 27.4.5 Ograniczenia metody stopniowego tworzenia oprogramowania
- 27.5 Podsumowanie
  - 27.5.1 Pytania kontrolne
    - 27.5.1.1 Metoda stopniowego tworzenia oprogramowania
  - 27.5.2 Warto przeczytać
    - 27.5.2.1 Integrowanie
    - 27.5.2.2 Stopniowe tworzenie oprogramowania
    - 27.5.2.3 Przyrostowe opracowywanie oprogramowania
  - 27.5.3 Warto zapamiętać

### **Rozdział 28 Strategie dopracowywania kodu źródłowego**

- 28.1 Ogólne informacje na temat wydajności

- 28.1.1 Jakość i wydajność
- 28.1.2 Wydajność i dopracowywanie kodu źródłowego
  - 28.1.2.1 Sposób zaprojektowania programu
  - 28.1.2.2 Sposób zaprojektowania modułów i podprogramów
  - 28.1.2.3 Współpraca z systemem operacyjnym
  - 28.1.2.4 Sposób kompilowania kodu źródłowego
  - 28.1.2.5 Sprzęt
  - 28.1.2.6 Kod źródłowy
- 28.2 Wprowadzenie do dopracowywania kodu źródłowego
  - 28.2.1 Legendy i mity
  - 28.2.2 Zasada Pareto
  - 28.2.3 Pomiary
    - 28.2.3.1 Wyniki pomiarów muszą być dokładne
  - 28.2.4 Optymalizacje wprowadzane przez kompilator
  - 28.2.5 Kiedy należy dopracowywać kod
  - 28.2.6 Iteracje
- 28.3 Ograniczenia wydajności
  - 28.3.1 Najczęstsze przyczyny niskiej wydajności
  - 28.3.2 Względna szybkość wykonywania typowych operacji
- 28.4 Podsumowanie wiadomości na temat dopracowywania kodu
- 28.5 Podsumowanie
  - 28.5.1 Warto przeczytać
    - 28.5.1.1 Wydajność
    - 28.5.1.2 Algorytmy i struktury danych
  - 28.5.2 Warto zapamiętać

## **Rozdział 29 Metody dopracowywania kodu źródłowego**

- 29.1 Pętle
  - 29.1.1 Usunięcie testów wykonywanych w pętlach
  - 29.1.2 Łączenie pętli
  - 29.1.3 Rozpisywanie
  - 29.1.4 Zmniejszanie liczby operacji wykonywanych wewnątrz pętli
  - 29.1.5 Wartości kontrolne
  - 29.1.6 Umieszczanie najbardziej czasochłonnych pętli wewnątrz
  - 29.1.7 Zastępowanie czasochłonnych działań
- 29.2 Wyrażenia logiczne
  - 29.2.1 Przerywanie sprawdzania warunku wtedy, gdy znamy jego wynik
  - 29.2.2 Uporządkowanie sprawdzanych warunków według prawdopodobieństwa, z jakim są one prawdziwe
  - 29.2.3 Tablice odpowiedzi zastępujące skomplikowane wyrażenia
  - 29.2.4 Wykonywanie obliczeń wtedy, gdy jest to potrzebne
- 29.3 Przekształcenia danych
  - 29.3.1 Korzystanie z liczb całkowitych zamiast z liczb zmiennoprzecinkowych
  - 29.3.2 Korzystanie z tablic o najmniejszych możliwych rozmiarach
  - 29.3.3 Zmniejszanie liczby operacji związanych z uzyskiwaniem dostępu do tablicy
  - 29.3.4 Korzystanie z pomocniczych indeksów
    - 29.3.4.1 Indeks długości napisu
    - 29.3.4.2 Indeks uporządkowanej jednokierunkowej listy wskaźników
    - 29.3.4.3 Niezależna struktura indeksów
  - 29.3.5 Korzystanie z buforowania
- 29.4 Wyrażenia
  - 29.4.1 Korzystanie z tożsamości algebraicznych
  - 29.4.2 Zastępowanie czasochłonnych działań
  - 29.4.3 Inicjalizowanie zmiennych przed skompilowaniem kodu
  - 29.4.4 Ostrożne korzystanie z podprogramów systemowych
  - 29.4.5 Korzystanie ze stałych o odpowiednich typach
  - 29.4.6 Wcześniejsze obliczanie potrzebnych wartości
  - 29.4.7 Usuwanie powtarzających się podwyrażeń
- 29.5 Podprogramy
  - 29.5.1 Umieszczanie kodu podprogramów bezpośrednio w programie

- 29.6 Ponowne napisanie podprogramu w assemblerze
- 29.7 Podsumowanie
  - 29.7.1 Warto przeczytać
  - 29.7.2 Warto zapamiętać

## **Rozdział 30 Rozwój oprogramowania**

- 30.1 Odmiany rozwoju oprogramowania
- 30.2 Ogólne wskazówki dotyczące rozwijania oprogramowania
  - 30.2.1 Filozofia rozwijania oprogramowania
- 30.3 Tworzenie nowych podprogramów
  - 30.3.1 Tworzenie nowych podprogramów w celu uproszczenia programu
    - 30.3.1.1 Upraszczenie programu w wyniku skracania podprogramów
    - 30.3.1.2 Upraszczenie programu w wyniku zmniejszenia liczby zagnieżdżonych poleceń
  - 30.3.2 Modyfikowanie podprogramów w celu wykorzystania istniejącego kodu
    - 30.3.2.1 Przykład współużytkowania kodu umieszczonego w podprogramie niskiego poziomu
    - 30.3.2.2 Przykład współużytkowania kodu umieszczonego w podprogramie wysokiego poziomu
- 30.4 Podsumowanie
  - 30.4.1 Pytania kontrolne
    - 30.4.1.1 Wprowadzanie zmian
  - 30.4.2 Warto przeczytać
  - 30.4.3 Warto zapamiętać

## **Część VIII**

### **Rozdział 31 Cechy charakteru programisty doskonałego**

- 31.1 Czy charakter należy do poruszanej tematyki
- 31.2 Inteligencja i skromność
- 31.3 Ciekawość
- 31.4 Rzetelność
- 31.5 Komunikatywność i współpraca
  - 31.5.1.1 Poziom 1. – początkujący
  - 31.5.1.2 Poziom 2. – średniozaawansowany
  - 31.5.1.3 Poziom 3. – specjalista
  - 31.5.1.4 Poziom 4. – ekspert
- 31.6 Kreatywność i dyscyplina
- 31.7 Lenistwo
- 31.8 Cechy charakteru, które mają mniejsze znaczenie niż się spodziewamy
  - 31.8.1 Wytrwałość
  - 31.8.2 Doświadczenie
  - 31.8.3 Programowanie w stylu macho
- 31.9 Nawyki
- 31.10 Podsumowanie
  - 31.10.1 Warto przeczytać
  - 31.10.2 Warto zapamiętać

### **Rozdział 32 Zagadnienia związane z tworzeniem oprogramowania**

- 32.1 Upraszczenie programu
  - 32.1.1 Metody upraszczania programu
  - 32.1.2 Hierarchie i stopień złożoności
  - 32.1.3 Poziom abstrakcji i złożoność
- 32.2 Wybieranie rodzaju procesu
- 32.3 Tworzenie programów dla ludzi, a dopiero potem dla komputerów
- 32.4 Korzystanie z konwencji
- 32.5 Programowanie z myślą o rozwiązywaniu problemów
  - 32.5.1 Dzielenie programu na fragmenty o różnych poziomach abstrakcji
    - 32.5.1.1 Poziom 1. – struktury języka wysokiego poziomu
    - 32.5.1.2 Poziom 2. – struktury informatyczne
    - 32.5.1.3 Poziom 3. – operacje niskiego poziomu dotyczące rozwiązywanego problemu
    - 32.5.1.4 Poziom 4. – operacje wysokiego poziomu dotyczące rozwiązywanego problemu

- 32.5.2 Metody niskiego poziomu przeznaczone do pracy nad problemem
- 32.6 Niebezpieczeństwa
- 32.7 Procesy iteracyjne
- 32.8 Oddzielenie procesu tworzenia oprogramowania od prywatnych przekonań
  - 32.8.1 Wyrocznie w dziedzinie programowania
  - 32.8.2 Eklektyzm
  - 32.8.3 Eksperymentowanie
- 32.9 Podsumowanie
  - 32.9.1 Warto zapamiętać

## **Rozdział 33 Gdzie można znaleźć więcej informacji**

- 33.1 Biblioteka programisty doskonałego
  - 33.1.1 Lista dziesięciu najprzydatniejszych książek
- 33.2 Informacje na temat tworzenia oprogramowania
- 33.3 Zagadnienia wykraczające poza tworzenie kodu
  - 33.3.1 Oprogramowanie w ogólności
  - 33.3.2 Przeglądy inżynierii oprogramowania
  - 33.3.3 Projektowanie interfejsu użytkownika
  - 33.3.4 Projektowanie baz danych
  - 33.3.5 Metody formalne
  - 33.3.6 Warto przeczytać
- 33.4 Czasopisma
  - 33.4.1 Popularne czasopisma dla programistów
  - 33.4.2 Czasopisma o charakterze teoretycznym przeznaczone dla programistów
  - 33.4.3 Publikacje przeznaczone dla wąskiego grona odbiorców
  - 33.4.4 Publikacje poświęcone konkretnym zagadnieniom
    - 33.4.4.1 Publikacje dla zawodowych programistów
    - 33.4.4.2 Publikacje przeznaczone dla szerszej grupy czytelników
- 33.5 Wstąpienie do organizacji zawodowej
- 33.6 Książki

## **Bibliografia**

## **Skorowidz**

## **Spis wydruków**