

Poznaj nowoczesną metodykę wytwarzania oprogramowania w C#

- Jak stosować w praktyce zasady zwinnego wytwarzania oprogramowania?
- W jaki sposób wykorzystywać w projekcie diagramy UML?
- Jak korzystać z wzorców projektowych?

PRENTICE  
HALL

# Agile Programowanie zwinne

Zasady, wzorce i praktyki zwinnego  
wytwarzania oprogramowania w

# C#

Helion 

Robert C. Martin, Micah Martin

Tytuł oryginału: Agile Principles, Patterns, and Practices in C#

Tłumaczenie: Mikołaj Szczepaniak

ISBN: 978-83-283-5567-5

Authorized translation from the English language edition, entitled: AGILE PRINCIPLES, PATTERNS, AND PRACTICES in C#, First Edition, ISBN 0131857258 by Robert C. Martin and Micah Martin, published by Pearson Education, Inc, publishing a Prentice Hall, Copyright © 2007 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 Wydawnictwo Helion S.A.,  
Copyright © 2008, 2019

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. 032 231 22 19, 032 230 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/agilev>  
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:  
<ftp://ftp.helion.pl/przyklady/agilev.zip>

Printed in Poland.

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

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

---

# Spis treści

---

Słowo wstępne .....	17
Przedmowa .....	21
Podziękowania .....	31
O autorach .....	33
<b>Część I Wytwarzanie zwinne .....</b>	<b>35</b>
<b>Rozdział 1. Praktyki programowania zwinnego .....</b>	<b>37</b>
<b>Agile Alliance .....</b>	<b>38</b>
Programiści i ich harmonijna współpraca jest ważniejsza od procesów i narzędzi .....	39
Działające oprogramowanie jest ważniejsze od wyczerpującej dokumentacji .....	40
Faktyczna współpraca z klientem jest ważniejsza od negocjacji zasad kontraktu .....	41
Reagowanie na zmiany jest ważniejsze od konsekwentnego realizowania planu .....	42
<b>Podstawowe zasady .....</b>	<b>43</b>
<b>Konkluzja .....</b>	<b>46</b>
<b>Bibliografia .....</b>	<b>47</b>
<b>Rozdział 2. Przegląd technik programowania     ekstremalnego .....</b>	<b>49</b>
<b>Praktyki programowania ekstremalnego .....</b>	<b>50</b>
Cały zespół .....	50
Opowieści użytkownika .....	50

Krótkie cykle .....	51
Testy akceptacyjne .....	52
Programowanie w parach .....	53
Wytwarzanie sterowane testami (TDD) .....	54
Wspólna własność .....	54
Ciągła integracja .....	55
Równe tempo .....	56
Otwarta przestrzeń pracy .....	56
Gra planistyczna .....	57
Prosty projekt .....	57
Refaktoryzacja .....	59
Metafora .....	59
<b>Konkluzja .....</b>	<b>61</b>
<b>Bibliografia .....</b>	<b>61</b>
<b>Rozdział 3. Planowanie .....</b>	<b>63</b>
<b>Wstępne poznawanie wymagań .....</b>	<b>64</b>
Dzielenie i scalanie opowieści użytkownika .....	65
<b>Planowanie wydań .....</b>	<b>66</b>
<b>Planowanie iteracji .....</b>	<b>66</b>
<b>Definiowanie warunków zakończenia projektu .....</b>	<b>67</b>
<b>Planowanie zadań .....</b>	<b>67</b>
<b>Iteracje .....</b>	<b>69</b>
<b>Śledzenie postępu .....</b>	<b>69</b>
<b>Konkluzja .....</b>	<b>70</b>
<b>Bibliografia .....</b>	<b>71</b>
<b>Rozdział 4. Testowanie .....</b>	<b>73</b>
<b>Wytwarzanie sterowane testami .....</b>	<b>74</b>
Przykład projektu poprzedzonego testami .....	75
Izolacja testów .....	76
Eliminowanie powiązań .....	78
<b>Testy akceptacyjne .....</b>	<b>79</b>
<b>Wpływ testów akceptacyjnych</b> <b>na architekturę oprogramowania .....</b>	<b>81</b>

<b>Konkluzja</b> .....	82
<b>Bibliografia</b> .....	82
<b>Rozdział 5. Refaktoryzacja</b> .....	83
<b>Prosty przykład refaktoryzacji</b>	
— <b>generowanie liczb pierwszych</b> .....	84
Testy jednostkowe .....	86
Refaktoryzacja .....	87
Ostatnie udoskonalenia .....	93
<b>Konkluzja</b> .....	97
<b>Bibliografia</b> .....	98
<b>Rozdział 6. Epizod z życia programistów</b> .....	99
<b>Gra w kręgle</b> .....	100
<b>Konkluzja</b> .....	146
<b>Przegląd reguł gry w kręgle</b> .....	147
<b>Część II Projektowanie zwinne</b> .....	149
<b>Rozdział 7. Czym jest projektowanie zwinne?</b> .....	153
<b>Symptomy złego projektu</b> .....	154
Symptomy złego projektu, czyli potencjalne źródła porażek .....	154
Szywność .....	155
Wrażliwość .....	155
Nieelastyczność .....	156
Niedostosowanie do rzeczywistości .....	156
Nadmierna złożoność .....	156
Niepotrzebne powtórzenia .....	157
Nieprzejrzystość .....	157
<b>Dlaczego oprogramowanie ulega degradacji</b> .....	158
<b>Program Copy</b> .....	159
Przykład typowego scenariusza .....	159
Przykład budowy programu Copy w ramach projektu zwinnego .....	163
<b>Konkluzja</b> .....	166
<b>Bibliografia</b> .....	166

<b>Rozdział 8. Zasada pojedynczej odpowiedzialności</b>	167
Definiowanie odpowiedzialności	170
Oddzielanie wzajemnie powiązanych odpowiedzialności	171
Trwałość	171
Konkluzja	172
Bibliografia	172
<b>Rozdział 9. Zasada otwarte-zamknięte</b>	173
Omówienie zasady otwarte-zamknięte	174
Aplikacja Shape	177
Przykład naruszenia zasady OCP	177
Przykład pełnej zgodności z zasadą otwarte-zamknięte	180
Przewidywanie zmian i „naturalna” struktura	181
Przygotowywanie punktów zaczepienia	182
Stosowanie abstrakcji do jawnego zamykania oprogramowania dla zmian	184
Zapewnianie zamknięcia z wykorzystaniem techniki sterowania przez dane	185
Konkluzja	187
Bibliografia	187
<b>Rozdział 10. Zasada podstawiania Liskov</b>	189
Naruszenia zasady podstawiania Liskov	190
Prosty przykład	190
Przykład mniej jaskrawego naruszenia zasady LSP	192
Przykład zaczerpnięty z rzeczywistości	199
Wyodrębnianie zamiast dziedziczenia	205
Heurystyki i konwencje	208
Konkluzja	209
Bibliografia	209
<b>Rozdział 11. Zasada odwracania zależności</b>	211
Podział na warstwy	212
Odwracanie relacji własności	213
Zależność od abstrakcji	215

Prosty przykład praktycznego znaczenia zasady DIP .....	216
Odkrywanie niezbędnych abstrakcji .....	217
Przykład aplikacji Furnace .....	219
Konkluzja .....	221
Bibliografia .....	221
<b>Rozdział 12. Zasada segregacji interfejsów .....</b>	<b>223</b>
Zanieczyszczanie interfejsów .....	223
Odrębne klasy klienckie oznaczają odrębne interfejsy .....	225
Interfejsy klas kontra interfejsy obiektów .....	227
Separacja przez delegację .....	227
Separacja przez wielokrotne dziedziczenie .....	229
Przykład interfejsu użytkownika bankomatu .....	230
Konkluzja .....	237
Bibliografia .....	237
<b>Rozdział 13. Przegląd języka UML</b>	
<b>dla programistów C# .....</b>	<b>239</b>
Diagramy klas .....	243
Diagramy obiektów .....	244
Diagramy sekwencji .....	245
Diagramy współpracy .....	246
Diagramy stanów .....	246
Konkluzja .....	247
Bibliografia .....	247
<b>Rozdział 14. Praca z diagramami .....</b>	<b>249</b>
<b>Po co modelować oprogramowanie? .....</b>	<b>249</b>
Po co budować modele oprogramowania? .....	250
Czy powinniśmy pracować nad rozbudowanymi projektami	
przed przystąpieniem do kodowania? .....	251
<b>Efektywne korzystanie z diagramów języka UML .....</b>	<b>251</b>
Komunikacja z innymi programistami .....	252
Mapy drogowe .....	253
Dokumentacja wewnętrzna .....	255
Co powinniśmy zachowywać, a co można wyrzucać do kosza? .....	255



<b>Iteracyjne udoskonalanie</b> .....	257
Najpierw zachowania .....	257
Weryfikacja struktury .....	259
Wyobrażenie o kodzie .....	261
Ewolucja diagramów .....	262
<b>Kiedy i jak rysować diagramy</b> .....	264
Kiedy przystępować do tworzenia diagramów, a kiedy rezygnować z dalszego rysowania ich .....	264
Narzędzia CASE .....	265
A co z dokumentacją? .....	266
<b>Konkluzja</b> .....	267
 <b>Rozdział 15. Diagramy stanów</b> .....	269
<b>Wprowadzenie</b> .....	270
Zdarzenia specjalne .....	271
Superstany .....	272
Pseudostan początkowy i końcowy .....	274
<b>Stosowanie diagramów skończonych maszyn stanów</b> .....	274
<b>Konkluzja</b> .....	276
 <b>Rozdział 16. Diagramy obiektów</b> .....	277
Migawka .....	278
<b>Obiekty aktywne</b> .....	279
<b>Konkluzja</b> .....	283
 <b>Rozdział 17. Przypadki użycia</b> .....	285
<b>Pisanie przypadków użycia</b> .....	286
Przebiegi alternatywne .....	287
Co jeszcze? .....	288
<b>Prezentowanie przypadków użycia na diagramach</b> .....	288
<b>Konkluzja</b> .....	290
<b>Bibliografia</b> .....	290
 <b>Rozdział 18. Diagramy sekwencji</b> .....	291
<b>Wprowadzenie</b> .....	292
Obiekty, linie życia, komunikaty i inne konstrukcje .....	292
Tworzenie i niszczenie obiektów .....	293



Proste pętle .....	295
Przypadki i scenariusze .....	295
<b>Pojęcia zaawansowane .....</b>	<b>298</b>
Pętle i warunki .....	298
Komunikaty, których przesyłanie wymaga czasu .....	300
Komunikaty asynchroniczne .....	302
Wiele wątków .....	307
Obiekty aktywne .....	308
Wysyłanie komunikatów do interfejsów .....	309
<b>Konkluzja .....</b>	<b>310</b>
<b>Rozdział 19. Diagramy klas .....</b>	<b>311</b>
<b>Wprowadzenie .....</b>	<b>312</b>
Klasy .....	312
Asocjacje .....	313
Relacje dziedziczenia .....	314
<b>Przykładowy diagram klas .....</b>	<b>315</b>
<b>Omówienie szczegółowe .....</b>	<b>318</b>
Stereotypy klas .....	318
Klasy abstrakcyjne .....	319
Właściwości .....	320
Agregacja .....	321
Kompozycja .....	322
Licznosc .....	323
Stereotypy asocjacji .....	324
Klasy zagnieżdżone .....	326
Klasy asocjacji .....	326
Kwalifikatory asocjacji .....	327
<b>Konkluzja .....</b>	<b>328</b>
Bibliografia .....	328
<b>Rozdział 20. Heurystyki i kawa .....</b>	<b>329</b>
<b>Ekspres do kawy Mark IV Special .....</b>	<b>330</b>
Specyfikacja .....	330
Popularne, ale niewłaściwe rozwiązanie .....	333
Nieprzemysłana abstrakcja .....	336
Poprawione rozwiązanie .....	337

Implementacja modelu abstrakcyjnego .....	343
Zalety projektu w tej formie .....	358
<b>Implementacja projektu obiektowego .....</b>	<b>366</b>
<b>Bibliografia .....</b>	<b>366</b>
<b>Część III Studium przypadku listy płac .....</b>	<b>367</b>
<b>Uproszczona specyfikacja systemu listy płac .....</b>	<b>368</b>
<b>Ćwiczenie .....</b>	<b>369</b>
Przypadek użycia nr 1 — dodanie danych nowego pracownika .....	369
Przypadek użycia nr 2 — usunięcie danych pracownika .....	370
Przypadek użycia nr 3 — wysłanie karty czasu pracy .....	370
Przypadek użycia nr 4 — wysłanie raportu o sprzedaży .....	370
Przypadek użycia nr 5 — wysłanie informacji o opłacie na rzecz związku zawodowego .....	371
Przypadek użycia nr 6 — zmiana szczegółowych danych pracownika ....	371
Przypadek użycia nr 7 — wygenerowanie listy płatności na dany dzień ..	372
<b>Rozdział 21. Wzorce projektowe Command i Active Object — uniwersalność i wielozadaniowość .....</b>	<b>373</b>
<b>Proste polecenia .....</b>	<b>374</b>
<b>Transakcje .....</b>	<b>377</b>
Fizyczny podział kodu .....	378
Czasowy podział kodu .....	379
<b>Metoda Undo .....</b>	<b>379</b>
<b>Wzorzec projektowy Active Object .....</b>	<b>380</b>
<b>Konkluzja .....</b>	<b>386</b>
<b>Bibliografia .....</b>	<b>386</b>
<b>Rozdział 22. Wzorce projektowe Template Method i Strategy — dziedziczenie kontra delegacja .....</b>	<b>387</b>
<b>Wzorzec projektowy Template Method .....</b>	<b>388</b>
Błędne zastosowanie wzorca Template Method .....	392
Sortowanie bąbelkowe .....	392

Wzorzec projektowy Strategy .....	396
Konkluzja .....	402
Bibliografia .....	402
<b>Rozdział 23. Wzorce projektowe Facade i Mediator .....</b>	<b>403</b>
Wzorzec projektowy Facade .....	404
Wzorzec projektowy Mediator .....	405
Konkluzja .....	407
Bibliografia .....	408
<b>Rozdział 24. Wzorce projektowe</b>	
<b>Singleton i Monostate .....</b>	<b>409</b>
Wzorzec projektowy Singleton .....	410
Zalety .....	412
Wady .....	412
Wzorzec Singleton w praktyce .....	413
Wzorzec projektowy Monostate .....	415
Zalety .....	417
Wady .....	417
Wzorzec Monostate w praktyce .....	417
Konkluzja .....	423
Bibliografia .....	423
<b>Rozdział 25. Wzorzec projektowy Null Object .....</b>	<b>425</b>
Omówienie .....	425
Konkluzja .....	429
Bibliografia .....	429
<b>Rozdział 26. Przypadek użycia listy płac</b>	
— pierwsza iteracja .....	431
Uproszczona specyfikacja .....	432
Analiza przez omówienie przypadku użycia .....	433
Dodanie danych nowego pracownika .....	434
Usunięcie danych pracownika .....	436
Wysyłanie karty czasu pracy .....	436
Wysyłanie raportu o sprzedaży .....	437

Wysłanie informacji o opłacie na rzecz związku zawodowego .....	438
Zmiana szczegółowych danych pracownika .....	439
Wygenerowanie listy płac na dany dzień .....	441
<b>Refleksja — identyfikacja abstrakcji</b> .....	443
Wynagrodzenia wypłacane pracownikom .....	444
Harmonogram wypłat .....	444
Formy wypłat .....	446
Przynależność związkowa .....	446
<b>Konkluzja</b> .....	447
<b>Bibliografia</b> .....	447

## Rozdział 27. Przypadek użycia listy płac

— <b>implementacja</b> .....	449
<b>Transakcje</b> .....	450
Dodawanie danych pracowników .....	450
Usuwanie danych pracowników .....	456
Karty czasu pracy, raporty o sprzedaży i składki na związki zawodowe .....	458
Zmiana danych pracowników .....	466
Co ja najlepszego zrobiłem? .....	477
Wynagradzanie pracowników .....	480
Wynagradzanie pracowników etatowych .....	483
Wynagradzanie pracowników zatrudnionych w systemie godzinowym .....	486
<b>Program główny</b> .....	498
<b>Baza danych</b> .....	499
<b>Konkluzja</b> .....	500
<b>O tym rozdziale</b> .....	501
<b>Bibliografia</b> .....	502

## Część IV Pakowanie systemu płacowego .....

### Rozdział 28. Zasady projektowania pakietów

<b>i komponentów</b> .....	505
Pakiety i komponenty .....	506
Zasady spójności komponentów — <b>ziarnistość</b> .....	507

Zasada równoważności wielokrotnego użycia i wydawania (REP) .....	507
Zasada zbiorowego wielokrotnego stosowania (CRP) .....	509
Zasada zbiorowego zamykania (CCP) .....	510
Podsumowanie problemu spójności komponentów .....	511
<b>Zasady spójności komponentów — stabilność</b> .....	<b>511</b>
Zasada acyklicznych zależności (ADP) .....	511
Zasada stabilnych zależności (SDP) .....	519
Zasada stabilnych abstrakcji (SAP) .....	525
<b>Konkluzja</b> .....	<b>530</b>
<b>Rozdział 29. Wzorzec projektowy Factory</b> .....	<b>531</b>
<b>Problem zależności</b> .....	<b>534</b>
<b>Statyczna kontra dynamiczna kontrola typów</b> .....	<b>536</b>
<b>Fabryki wymienne</b> .....	<b>537</b>
<b>Wykorzystywanie fabryk do celów testowych</b> .....	<b>538</b>
<b>Znaczenie fabryk</b> .....	<b>540</b>
<b>Konkluzja</b> .....	<b>540</b>
<b>Bibliografia</b> .....	<b>540</b>
<b>Rozdział 30. Studium przypadku systemu płacowego</b>	
<b>— analiza podziału na pakiety</b> .....	<b>541</b>
<b>Notacja i struktura komponentów</b> .....	<b>542</b>
<b>Stosowanie zasady zbiorowego zamykania (CCP)</b> .....	<b>544</b>
<b>Stosowanie zasady równoważności wielokrotnego użycia</b> <b>    i wydawania (REP)</b> .....	<b>546</b>
<b>Wiązanie komponentów i hermetyzacja</b> .....	<b>549</b>
<b>Mierniki</b> .....	<b>551</b>
<b>Stosowanie mierników dla aplikacji płacowej</b> .....	<b>553</b>
<b>Fabryki obiektów</b> .....	<b>556</b>
<b>Przebudowa granic spójności</b> .....	<b>558</b>
<b>Ostateczna struktura pakietów</b> .....	<b>559</b>
<b>Konkluzja</b> .....	<b>561</b>
<b>Bibliografia</b> .....	<b>561</b>

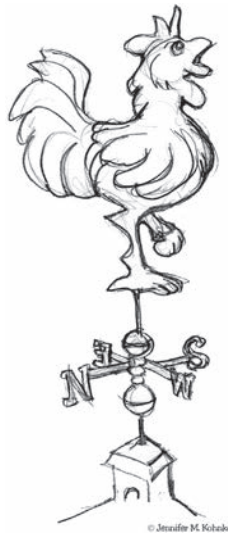
<b>Rozdział 31. Wzorzec projektowy Composite</b> .....	563
Polecenia kompozytowe .....	565
Licznosc albo brak licznosci .....	566
Konkluzja .....	566
<b>Rozdział 32. Wzorzec projektowy Observer</b>	
— ewolucja kodu w kierunku wzorca .....	567
Zegar cyfrowy .....	568
Wzorzec projektowy Observer .....	589
Modele .....	590
Zarządzanie zasadami projektowania obiektowego .....	591
Konkluzja .....	592
Bibliografia .....	593
<b>Rozdział 33. Wzorce projektowe Abstract Server,</b>	
<b>Adapter i Bridge</b> .....	595
Wzorzec projektowy Abstract Server .....	596
Wzorzec projektowy Adapter .....	598
Forma klasowa wzorca Adapter .....	599
Problem modemu — adaptery i zasada LSP .....	599
Wzorzec projektowy Bridge .....	604
Konkluzja .....	607
Bibliografia .....	607
<b>Rozdział 34. Wzorce projektowe Proxy i Gateway</b>	
— zarządzanie cudzymi interfejsami API ....	609
Wzorzec projektowy Proxy .....	610
Implementacja wzorca Proxy .....	615
Podsumowanie .....	630
<b>Bazy danych, oprogramowanie pośredniczące</b>	
<b>i inne gotowe interfejsy</b> .....	631
Wzorzec projektowy Table Data Gateway .....	634
Testowanie konstrukcji TDG w pamięci .....	642
Test bram DB .....	643

Stosowanie pozostałych wzorców projektowych łącznie z bazami danych .....	646
Konkluzja .....	648
Bibliografia .....	648
<b>Rozdział 35. Wzorzec projektowy Visitor .....</b>	<b>649</b>
Wzorzec projektowy Visitor .....	650
Wzorzec projektowy Acyclic Visitor .....	654
Zastosowania wzorca Visitor .....	660
Wzorzec projektowy Decorator .....	668
Wzorzec projektowy Extension Object .....	674
Konkluzja .....	686
Bibliografia .....	686
<b>Rozdział 36. Wzorzec projektowy State .....</b>	<b>687</b>
Zagnieżdżone wyrażenia switch-case .....	688
Wewnętrzny zasięg zmiennej stanu .....	691
Testowanie akcji .....	692
Zalety i wady .....	692
Tabele przejść .....	693
Interpretacja tabeli przejść .....	694
Zalety i wady .....	695
Wzorzec projektowy State .....	696
Wzorzec State kontra wzorzec Strategy .....	699
Zalety i wady .....	699
Kompilator maszyny stanów (SMC) .....	700
Plik Turnstile.cs wygenerowany przez kompilator SMC i pozostałe pliki pomocnicze .....	703
Zastosowania skończonej maszyny stanów .....	709
Wysokopoziomowa polityka działania graficznych interfejsów użytkownika (GUI) .....	709
Sterowanie interakcją z interfejsem GUI .....	711
Przetwarzanie rozproszone .....	712
Konkluzja .....	713
Bibliografia .....	714



<b>Rozdział 37. Studium przypadku systemu płacowego</b>	
— baza danych .....	715
Budowa bazy danych .....	716
Słaby punkt dotychczasowego projektu .....	716
Dodawanie danych nowych pracowników .....	719
Transakcje .....	732
Odczytywanie danych o pracownikach .....	738
Co jeszcze zostało do zrobienia? .....	753
<b>Rozdział 38. Interfejs użytkownika systemu płacowego</b>	
— wzorzec Model View Presenter .....	755
Interfejs .....	758
Implementacja .....	759
Budowa okna .....	770
Okno główne systemu płacowego .....	778
Pierwsza odsłona .....	791
Konkluzja .....	792
Bibliografia .....	792
<b>Dodatek A Satyra na dwa przedsiębiorstwa</b> .....	793
Rufus Inc. — Project Kickoff .....	793
Rupert Industries — Project Alpha .....	793
<b>Dodatek B Czym jest oprogramowanie?</b> .....	811
<b>Skorowidz</b> .....	827

## Praktyki programowania zwinnego



*Kogut na wieży kościelnej, choć wykuty z żelaza, szybko zostałby strącony, gdyby choć na moment zapomniał o powinności ulegania najmniejszemu powiewom.*

— Heinrich Heine

Wielu z nas przeżyło koszmar pracy nad projektem, dla którego nie istniał zbiór sprawdzonych praktyk, jakimi można by się kierować. Brak efektywnych praktyk prowadzi do nieprzewidywalności, wielokrotnych błędów i niepotrzebnie podejmowanych działań.

Klienci są niezadowoleni z niedotrzymywanych terminów, rosnących kosztów i kiepskiej jakości. Sami programiści są zniechęceni ciągłymi nadgodzinami i świadomością niskiej jakości budowanego oprogramowania.

Po doświadczeniu podobnej sytuacji stajemy się ostrożniejsi, aby podobne fiasko nigdy nie stało się ponownie naszym udziałem. Tego rodzaju obawy często okazują się dobrą motywacją do definiowania procesów determinujących kolejne działania (włącznie z ich wynikami i produktami pośrednimi). Ograniczenia i oczekiwania względem poszczególnych etapów tych procesów definiujemy na podstawie dotychczasowych doświadczeń — z reguły decydujemy się na rozwiązania, które zdawały egzamin w przeszłości. Każdy taki wybór wiąże się z przeświadczeniem, że odpowiednie rozwiązania okażą się skuteczne raz jeszcze.

Projekty nie są jednak na tyle proste, by zaledwie kilka ograniczeń i gotowych komponentów mogło nas uchronić przed błędami. Musimy się liczyć z dalszym występowaniem niedociągnięć, których diagnozowanie skłania nas do definiowania kolejnych ograniczeń i oczekiwanych produktów końcowych, jakie w założeniu mają nas uchronić przed podobnymi błędami w przyszłości. Po wielu realizowanych w ten sposób projektach może się okazać, że nasze metody działania są skomplikowane, niewygodne i w praktyce uniemożliwiają efektywną realizację projektów.

Skomplikowany i nieusprawniający pracy proces może prowadzić do wielu problemów, które za pomocą tych metod chcieliśmy wyeliminować. Skutki stosowania takich procesów mogą obejmować opóźnienia czasowe i przekroczenia zakładanych wydatków budżetowych. Rozmaite niejasności w ramach tych procesów mogą rozmywać odpowiedzialność zespołu i — tym samym — prowadzić do powstawania niedopracowanych produktów. Co ciekawe, opisywane zjawiska skłaniają wiele zespołów do przekonania, że źródłem problemów jest niedostateczna liczba procesów. W takim przypadku odpowiedzią na negatywne skutki nadmiernej liczby przesadnie rozrośniętych procesów jest pogłębianie tych zjawisk.

Nadmierne rozbudowywanie procesów było powszechnym zjawiskiem w wielu firmach budujących oprogramowanie koło 2000 roku. Mimo że jeszcze w 2000 roku wiele zespołów tworzyło oprogramowanie bez procesów, popularyzacja bardzo skomplikowanych procesów postępowała w błyskawicznym tempie (szczególnie w największych korporacjach).

## Agile Alliance

Grupa ekspertów zaniepokojonych obserwowanymi zjawiskami (polegającymi między innymi na wpędzaniu się wielu korporacji w poważne kłopoty wskutek rozrastających się procesów) zorganizowała na początku 2001 roku spotkanie, na którym powstało zrzeszenie nazwane Agile Alliance. Celem tej grupy była popularyzacja wartości i reguł, które skłonią zespoły programistyczne do szybkiego i elastycznego wytwarzania oprogramowa-

nia. Przez kilka miesięcy grupie Agile Alliance udało się opracować najważniejsze założenia. Efektem tej pracy był dokument zatytułowany *The Manifesto of the Agile Alliance*:

### Manifest zwinnego wytwarzania oprogramowania

*Próbujemy odkrywać lepsze sposoby wytwarzania oprogramowania, realizując własne eksperymenty w tym obszarze i zachęcając do podobnych działań innych programistów. Wskutek tych eksperymentów udało nam się sformułować następujące wskazówki:*

**Programiści i ich harmonijna współpraca** jest ważniejsza od procesów i narzędzi.

**Działające oprogramowanie** jest ważniejsze od wyczerpującej dokumentacji.

**Faktyczna współpraca z klientem** jest ważniejsza od negocjacji zasad kontraktu.

**Reagowanie na zmiany** jest ważniejsze od konsekwentnego realizowania planu.

*Oznacza to, że chociaż rozwiązania wymienione po prawej stronie mają pewną wartość, z punktu widzenia skuteczności projektów dużo bardziej wartościowe są działania opisane z lewej strony.*

Kent Beck	Mike Beedle	Arie van Bennekum	Alistair Cockburn
Ward Cunningham	Martin Fowler	James Grenning	Jim Highsmith
Andrew Hunt	Ron Jeffries	Jon Kern	Brian Marick
Robert C. Martin	Steve Mellor	Ken Schwaber	Jeff Sutherland
Dave Thomas			

## Programiści i ich harmonijna współpraca jest ważniejsza od procesów i narzędzi

Ludzie zawsze są najważniejszym czynnikiem decydującym o sukcesie. Żaden, nawet najlepszy proces nie zapobiegnie porażce projektu, jeśli nie mamy do dyspozycji zespołu, któremu możemy zaufać. Co więcej, zły proces może spowodować, że nawet najlepszy zespół będzie pracował nieefektywnie. Okazuje się, że grupa doskonałych programistów może ponieść dotkliwą porażkę, jeśli nie stworzy prawdziwego zespołu.

Dobry członek zespołu wcale nie musi być genialnym programistą. Dobry pracownik może być przeciętnym programistą, ale musi posiadać zdolność współpracy z pozostałymi członkami swojego zespołu. Harmonijna współpraca z innymi pracownikami i umiejętność komunikacji jest ważniejsza od talentów czysto programistycznych. Prawdopodobieństwo odniesienia sukcesu przez grupę przeciętnych programistów, którzy dobrze ze sobą współpracują, jest większe niż w przypadku zespołu złożonego z samych geniuszy programowania niepotrafiących się porozumieć.

Dobrze dobrane narzędzia mogą mieć istotny wpływ na ostateczny efekt realizacji projektu. Kompilatory, interaktywne środowiska wytwarzania (IDE), systemy kontroli wersji kodu źródłowego itp. są kluczowymi narzędziami decydującymi o funkcjonowaniu

zespołu programistów. Z drugiej strony, nie należy przeceniać roli samych narzędzi. Nadmierna wiara w siłę narzędzi prowadzi do równie fatalnych skutków co ich brak.

Zaleca się stosowanie z początku stosunkowo prostych narzędzi. Nie należy zakładać, że bardziej rozbudowane i zaawansowane narzędzia będą lepsze od prostych, do czasu weryfikacji ich przydatności w praktyce. Zamiast kupować najlepsze i niewyobrażalnie drogie systemy kontroli wersji kodu źródłowego, warto poszukać darmowych narzędzi tego typu i korzystać z nich do czasu, aż wyczerpią się oferowane przez nie możliwości. Zanim kupimy najdroższy pakiet narzędzi CASE (od ang. *Computer-Aided Software Engineering*), powinniśmy skorzystać ze zwykłych tablic i kartek papieru — decyzję o zakupie bardziej zaawansowanych rozwiązań możemy podjąć dopiero wtedy, gdy tradycyjne metody okażą się niewystarczające. Przed podjęciem decyzji o zakupie niezwykle drogiego i zaawansowanego systemu zarządzania bazą danych powinniśmy sprawdzić, czy do naszych celów nie wystarczą zwykłe pliki. Nigdy nie powinniśmy zakładać, że większe i lepsze narzędzia muszą w większym stopniu odpowiadać naszym potrzebom. Wiele z nich bardziej utrudnia, niż ułatwia pracę.

Warto pamiętać, że budowa zespołu jest ważniejsza od budowy środowiska. Wiele zespołów i wielu menedżerów popełnia błąd polegający na budowie w pierwszej kolejności środowiska w nadziei, że wokół niego uda się następnie zebrać zgrany zespół. Dużo lepszym rozwiązaniem jest skoncentrowanie się na budowie zespołu i umożliwienie mu organizacji środowiska odpowiadającego jego potrzebom.

## Działające oprogramowanie jest ważniejsze od wyczerpującej dokumentacji

Oprogramowanie bez dokumentacji jest prawdziwą katastrofą. Kod źródłowy nie jest jednak idealnym medium komunikowania przyczyn stosowania poszczególnych rozwiązań i struktury systemu. Lepszym sposobem jest uzasadnianie podejmowanych przez zespół programistów decyzji projektowych w formie zwykłych dokumentów.

Z drugiej strony, zbyt duża liczba dokumentów jest gorsza od zbyt skromnej dokumentacji. Wytwarzanie wielkich dokumentów opisujących oprogramowanie wymaga dużych nakładów czasowych, a w skrajnych przypadkach uniemożliwia synchronizację dokumentacji z kodem źródłowym. Dokumentacja niezgodna z faktycznym stanem kodu staje się wielkim, złożonym zbiorem kłamstw, które prowadzą tylko do nieporozumień.

Niezależnie od realizowanego projektu dobrym rozwiązaniem jest pisanie i utrzymywanie przez zespół programistów krótkiego dokumentu uzasadniającego podjęte decyzje projektowe i opisującego strukturę budowanego systemu. Taki dokument powinien być możliwie **krótki** i dość ogólny — nie powinien zajmować więcej niż dwadzieścia stron i powinien dotyczyć **najważniejszych** decyzji projektowych oraz opisywać strukturę systemu na najwyższym poziomie.

Skoro dysponujemy jedynie krótkim dokumentem uzasadniającym decyzje projektowe i strukturę budowanego systemu, jak będziemy przygotowywać do pracy nad tym systemem nowych członków zespołu? Taki trening nie powinien polegać na przeka-

zaniu dokumentów, tylko na żmudnej pracy z nowym programistą. Transfer wiedzy o systemie musi polegać na wielogodzinnej pomocy nowemu współpracownikowi. Tylko blisko współpracując z programistą, możemy z niego szybko uczynić pełnowartościowego członka naszego zespołu.

Najlepszymi mediami umożliwiającymi przekazanie informacji niezbędnych do pracy nowych członków zespołu jest kod źródłowy i sam zespół. Kod nigdy nie jest sprzeczny ze swoim faktycznym działaniem. Z drugiej strony, mimo że precyzyjne określenie funkcjonowania systemu na podstawie samego kodu źródłowego może być trudne, właśnie kod jest jedynym jednoznacznym źródłem informacji. Zespół programistów utrzymuje swoistą mapę drogową stale modyfikowanego systemu w głowach swoich członków. Oznacza to, że najszybszym i najbardziej efektywnym sposobem prezentacji założeń systemu osobom spoza zespołu jest przelanie tej mapy na papier.

Wiele zespołów wpadło w pułapkę pogoni za doskonałą dokumentacją kosztem właściwego oprogramowania. Takie podejście prowadzi do fatalnych skutków. Można tego błędu uniknąć, stosując następującą regułę:

**Pierwsze prawo Martina dotyczące dokumentacji**

*Nie pracuj nad żadnymi dokumentami, chyba że w danym momencie bardzo ich potrzebujesz.*

## Faktyczna współpraca z klientem jest ważniejsza od negocjacji zasad kontraktu

Oprogramowanie nie może być zamawiane jak zwykły towar oferowany w sklepie. Opisanie potrzebnego oprogramowania i znalezienie kogoś, kto je dla nas opracuje w określonym terminie i za ustaloną cenę, jest po prostu niemożliwe. Wielokrotnie podejmowane próby traktowania projektów informatycznych w ten sposób zawsze kończyły się niepowodzeniem. Część tych niepowodzeń była dość spektakularna.

Kadra menedżerska często ulega pokusie przekazania zespołowi programistów swoich potrzeb z myślą o odbyciu kolejnego spotkania dopiero wtedy, gdy wrócą z gotowym systemem (w pełni zgodnym z oczekiwaniami zamawiających). Ten tryb współpracy prowadzi jednak albo do tworzenia oprogramowania fatalnej jakości, albo do zupełnego niepowodzenia projektów.

Warunkiem powodzenia projektu jest stała współpraca z klientem, najlepiej w formie regularnych, możliwie częstych spotkań. Zamiast opierać się wyłącznie na zapisach kontraktu, zespół programistów może ściśle współpracować z klientem, aby obie strony stale miały świadomość wzajemnych potrzeb i ograniczeń.

Kontrakt precyzujący wymagania, harmonogram i koszty przyszłego systemu z natury rzeczy musi zawierać poważne błędy. W większości przypadków zapisy takiego kontraktu stają się nieaktualne na długo przed zakończeniem realizacji projektu, a w skrajnych

przypadkach dezaktualizują się przed jego podpisaniem! Najlepsze kontrakty to takie, które przewidują ścisłą współpracę zespołu programistów z przedstawicielami klienta.

W 1994 roku miałem okazję negocjować kontrakt dla wielkiego, wieloletniego projektu (obejmującego pół miliona wierszy kodu), który zakończył się pełnym sukcesem. Członkowie zespołu programistów, do którego należałem, otrzymywali stosunkowo niewielkie uposażenia miesięczne i całkiem spore premie po dostarczeniu każdego większego bloku funkcjonalności. Co ciekawe, tego rodzaju bloki nie były szczegółowo opisane we wspomnianym kontrakcie. Umowa gwarantowała, że premie będą wypłacane dopiero wtedy, gdy dany blok przejdzie pozytywnie testy akceptacyjne klienta. Kontrakt nie określał też szczegółów przeprowadzania samych testów akceptacyjnych, których przebieg zależał wyłącznie od klienta zainteresowanego jak najlepszą jakością produktu końcowego.

W czasie realizacji tego projektu bardzo blisko współpracowaliśmy z klientem. Niemal w każdy piątek przekazywaliśmy mu jakąś część budowanego oprogramowania. Już w poniedziałek lub we wtorek kolejnego tygodnia otrzymywaliśmy listę zmian oczekiwanych przez klienta. Wspólnie nadawaliśmy tym modyfikacjom priorytety, po czym planowaliśmy ich wprowadzanie w ciągu kolejnych tygodni. Nasza współpraca z przedstawicielami klienta była na tyle bliska, że testy akceptacji nigdy nie stanowiły problemu. Doskonale wiedziliśmy, kiedy poszczególne bloki funkcjonalności spełniały oczekiwania klienta, ponieważ jego przedstawiciele stale obserwowali postępy prac.

Wymagania stawiane temu projektowi stale były modyfikowane. Bardzo często decydowaliśmy się na wprowadzanie bardzo istotnych zmian. Zdarzało się, że rezygnowaliśmy lub dodawaliśmy całe bloki funkcjonalności. Mimo to zarówno kontrakt, jak i sam projekt okazał się pełnym sukcesem. Kluczem do tego sukcesu była intensywna współpraca z klientem i zapisy kontraktu, które nie definiowały szczegółowego zakresu, harmonogramu ani kosztów, tylko regulowały zasady współpracy wykonawcy ze zleceniodawcą.

## **Reagowanie na zmiany jest ważniejsze od konsekwentnego realizowania planu**

O sukcesie bądź porażce projektu polegającego na budowie oprogramowania decyduje zdolność częstego i szybkiego reagowania na zmiany. Pracując nad planami realizacji projektu, koniecznie musimy się upewnić, że tworzony harmonogram będzie na tyle elastyczny, aby można było w jego ramach wprowadzać zmiany (zarówno w wymiarze biznesowym, jak i technologicznym).

Projektowi polegającego na tworzeniu systemu informatycznego nie można szczegółowo zaplanować. Po pierwsze, musimy się liczyć ze zmianami otoczenia biznesowego, które będą miały wpływ na stawiane nam wymagania. Po drugie, kiedy nasz system wreszcie zacznie działać, możemy być niemal pewni, że przedstawiciele klienta złączą wspominać o zmianach dotychczasowych wymagań. I wreszcie, nawet jeśli mamy pełną wiedzę o wymaganiach i jesteśmy pewni, że nie będą się zmieniały, nie jesteśmy w stanie precyzyjnie oszacować czasu ich realizacji.



Mniej doświadczeni menedżerowie ulegają pokusie nanoszenia całego projektu na wielkie arkusze papieru i rozklejania ich na ścianach. Wydaje im się, że takie schematy dają im pełną kontrolę nad projektem. Mogą łatwo śledzić wybrane zadania i wykreślać je ze schematu zaraz po ich wykonaniu. Mogą też porównywać obserwowane daty realizacji zadań z datami planowanymi i właściwie reagować na wszelkie odstępstwa od harmonogramu.

**Prawdziwym** problemem jest jednak to, że struktura takiego schematu szybko się dezaktualizuje. Wraz ze wzrostem wiedzy zespołu programistów na temat budowanego systemu rośnie wiedza klienta o potrzebach samego zespołu, a część zadań reprezentowanych na wykresie okazuje się po prostu zbędna. W tym samym czasie mogą być odkrywane zupełnie inne zadania, które należy dodać do istniejącego schematu. Krótko mówiąc, **zmianie** ulegają nie tylko daty realizacji poszczególnych zadań, ale także same zadania.

Dużo lepszą strategią planowania jest tworzenie szczegółowych harmonogramów i wykazów zadań na najbliższy tydzień, ogólnych planów na kolejne trzy miesiące oraz bardzo ogólnych, wręcz szkicowych planów na dalsze okresy. Powinniśmy mieć precyzyjną wiedzę o zadaniach, nad którymi będziemy pracować w przyszłym tygodniu, mniej więcej znać wymagania, jakie będziemy realizować w ciągu kolejnych trzech miesięcy, oraz mieć jakieś wyobrażenie o możliwym kształcie naszego systemu po roku.

Stopniowo obniżany poziom szczegółowości formułowanych planów oznacza, że poświęcamy czas na szczegółowe szacowanie tylko tych zadań, którymi będziemy się zajmowali w najbliższej przyszłości. Modyfikowanie raz opracowanego, szczegółowego planu działań jest trudne, bo w jego tworzenie i zatwierdzenie był zaangażowany cały zespół. Ponieważ jednak taki plan dotyczy tylko najbliższego tygodnia, pozostałe szacunki pozostają elastyczne.

## Podstawowe zasady

Na podstawie ogólnych reguł opisanych w poprzednim podrozdziale można sformułować następujący zbiór dwunastu szczegółowych zasad. Właśnie te reguły odróżniają praktyki programowania zwinnego od tradycyjnych, ciężkich procesów:

1. **Naszym najwyższym nakazem jest spełnienie oczekiwań klienta przez możliwie wczesne dostarczanie wartościowego oprogramowania.** W kwartalniku „MIT Sloan Management Review” opublikowano kiedyś analizę praktyk wytwarzania oprogramowania, które ułatwiają przedsiębiorstwom budowę produktów wysokiej jakości<sup>3</sup>. W przytoczonym artykule zidentyfikowano szereg różnych praktyk, które miały istotny wpływ na jakość ostatecznej wersji systemu. Opisano między innymi ścisłą korelację pomiędzy jakością końcowego produktu a wczesnym dostarczeniem częściowo funkcjonującego systemu. W artykule dowiedziono, że **im mniej**

---

<sup>3</sup> *Product-Development Practices That Work: How Internet Companies Build Software*, „MIT Sloan Management Review”, zima 2001, nr 4226.

**funkcjonalny będzie początkowo dostarczony fragment systemu, tym wyższa będzie jakość jego wersji końcowej.** W tym samym artykule zwrócono też uwagę na związek pomiędzy jakością systemu końcowego a częstotliwością dostarczania klientowi coraz bardziej funkcjonalnych wersji pośrednich. **Im częściej przekazujemy klientowi gotowy fragment funkcjonalności, tym wyższa będzie jakość produktu końcowego.**

Zbiór praktyk programowania zwinnego nakazuje nam możliwie wczesne i częste dostarczanie fragmentów oprogramowania. Powinniśmy przekazać klientowi początkową wersję budowanego systemu już w ciągu kilku tygodni od rozpoczęcia prac nad projektem. Od momentu przekazania pierwszej, najskromniejszej wersji oprogramowania powinniśmy co kilka tygodni dostarczać klientowi coraz bardziej funkcjonalne warianty. Jeśli klient uzna, że zaoferowana funkcjonalność jest wystarczająca, może się zdecydować na wdrożenie pośredniej wersji produktu w środowisku docelowym. W przeciwnym razie klient dokonuje przeglądu istniejącej funkcjonalności i przekazuje wykonawcy wykaz oczekiwanych zmian.

- 2. Traktujmy zmiany wymagań ze zrozumieniem, nawet jeśli następują w późnych fazach wytwarzania.** Procesy zwinne muszą się zmieniać wraz ze zmieniającymi się uwarunkowaniami biznesowymi, w których funkcjonuje strona zamawiająca. Wszystko jest kwestią właściwego podejścia. Uczestnicy procesów zwinnych z reguły nie obawiają się zmian — uważają je wręcz za zjawisko pozytywne, które prowadzi do lepszego rozumienia przez zespół programistów oczekiwań klienta.

Zespół pracujący nad zwinnym projektem musi poświęcać wiele czasu na utrzymywanie właściwej elastyczności struktury swojego oprogramowania, ponieważ tylko takie podejście pozwala zminimalizować wpływ zmian wymagań na kształt budowanego systemu. W dalszej części tej książki omówimy reguły, wzorce projektowe i praktyki programowania obiektowego, które ułatwiają nam zachowywanie takiej elastyczności.

- 3. Działające oprogramowanie należy dostarczać klientowi możliwie często (w odstępach kilkutygodniowych lub kilkumiesięcznych).** Powinniśmy przekazywać stronie zamawiającej pierwszą wersję pracującego oprogramowania tak szybko, jak to tylko możliwe, i kontynuować ten proces na wszystkich etapach realizacji projektu. Nie wystarczy dostarczanie plików dokumentów ani nawet najbardziej rozbudowanych planów. Żaden dokument nie zastąpi rzeczywistej funkcjonalności przekazywanej klientowi. Naszym ostatecznym celem jest dostarczenie zamawiającemu oprogramowania, które w pełni będzie spełniało jego oczekiwania.
- 4. Ludzie biznesu powinni ściśle współpracować z programistami na wszystkich etapach projektu.** Warunkiem zwinności projektu jest częsta i intensywna współpraca klientów, programistów i ośrodków biznesowych zaangażowanych w jego

finansowanie. Projektu polegającego na tworzeniu oprogramowania nie można traktować jak broni „odpal i zapomnij”. Tego rodzaju projekty muszą być stale kierowane.

5. **Projekty należy planować wokół dobrze umotywowanych programistów. Należy im zorganizować niezbędne środowisko i wsparcie, a także obdarzyć ich potrzebnym zaufaniem.** To ludzie są najważniejszym czynnikiem decydującym o powodzeniu projektu. Inne czynniki (w tym proces, środowisko i sposób zarządzania) mają znaczenie drugorzędne i mogą być dowolnie dostosowywane do potrzeb ludzi zaangażowanych w realizację projektu.
6. **Najbardziej efektywną metodą przekazywania informacji do i w ramach zespołu programistów jest rozmowa w cztery oczy.** Projekty zwinne polegają w dużej mierze właśnie na rozmowach prowadzonych przez członków zespołu. Podstawowym trybem takiej komunikacji są bezpośrednie kontakty programistów. Ewentualne dokumenty są pisane i aktualizowane równoległe do takich rozmów (zgodnie z harmonogramem prac nad oprogramowaniem).
7. **Podstawowym miernikiem postępu prac nad projektem jest ilość i jakość działającego oprogramowania.** Miarą postępu projektu zwinnego jest ilość oprogramowania, która w danej chwili spełnia oczekiwania klienta. Szacowanie bieżącego stanu prac w żadnym razie nie powinno polegać na weryfikacji realizowanej fazy projektu, objętości wytworzonej dokumentacji czy ilości gotowego kodu infrastrukturalnego. O 30% postępie możemy mówić tylko wtedy, gdy 30% funkcjonalności działa i spotyka się z akceptacją klienta.
8. **Procesy zwinne ułatwiają utrzymywanie ciągłości wytwarzania. Sponsorzy, programiści i użytkownicy powinni mieć możliwość utrzymywania stałego tempa pracy przez cały czas realizacji projektu.** Projekt zwinny dużo bardziej przypomina maraton niż bieg sprinterski na 100 metrów. Zespół programistów nie powinien od razu próbować osiągnąć maksymalnej wydajności, której utrzymanie w dłuższym okresie byłoby niemożliwe. Lepszym rozwiązaniem jest utrzymywanie wysokiej, ale stałej szybkości pracy.

Narzucanie zbyt wysokiego tempa pracy prowadzi do wypalenia psychicznego, podejmowania decyzji o wyborze rozmaitych skrótów i ostatecznego fiaska. Zwinne zespoły potrafią same utrzymywać stałą wydajność — żaden członek takiego zespołu nie musi korzystać z zapasów jutrzejszej energii z myślą o dłuższej bądź szybszej pracy dzisiaj. Programiści wchodzący w skład zwinnych zespołów utrzymują stałe tempo prac, które umożliwia im realizację standardów najwyższej jakości przez cały okres trwania prac nad realizacją projektu.
9. **Pozytywny wpływ na zwinność projektu ma stała dbałość o doskonałość techniczną i właściwy projekt.** Wysoka jakość jest kluczem do dużej wydajności. Oznacza to, że właściwym sposobem zachowania zadowalającej szybkości jest utrzymywanie możliwie prostej i przemyślanej struktury oprogramowania. Wszyscy

członkowie zwinnego zespołu powinni się koncentrować na wytwarzaniu kodu źródłowego o najwyższej jakości, na jaką pozwalają ich umiejętności. Nie doprowadzają do sytuacji, w której muszą kogoś zapewniać, że zaległe zadania zrealizują w bliżej nieokreślonej przyszłości. Eliminują źródła potencjalnych opóźnień w zarodku.

10. **Kluczowym elementem pomyslnego zakończenia projektu jest prostota, czyli sztuka maksymalizacji ilości pracy, której nie wykonujemy.** Zwinne zespoły nie próbują na siłę budować wielkich i skomplikowanych systemów — zawsze starają się dochodzić do stawianych im celów najkrótszą drogą. Nie przywiązują zbyt dużej wagi do ewentualnych problemów dnia jutrzejszego ani nie próbują za wszelką cenę zmagać się z problemami bliższymi aktualnie realizowanym zadaniom. Ich celem jest osiągnięcie możliwie najwyższej jakości z jednoczesnym zapewnieniem elastyczności umożliwiającej łatwe modyfikowanie oprogramowania w odpowiedzi na przyszłe problemy.
11. **Źródłem najlepszych architektur, specyfikacji wymagań i projektów są zespoły, którym dano wolną rękę w zakresie organizacji.** Zwinny zespół programistów musi sam organizować swoją pracę. Odpowiedzialność za poszczególne zadania nie powinna być przydzielana wybranym członkom zespołu z zewnątrz, tylko komunikowana całemu zespołowi. Sam zespół powinien następnie decydować, jak te zadania najefektywniej zrealizować.

Członkowie zwinnego zespołu wspólnie pracują nad wszystkimi aspektami zleconego projektu. Każdy członek takiego zespołu może zgłaszać swoje propozycje dotyczące dowolnego aspektu realizowanego zlecenia. Za takie zadania, jak tworzenie architektury, definiowanie wymagań czy testy, nigdy nie odpowiadają poszczególni programiści. Odpowiedzialność za tak kluczowe czynności musi obejmować cały zespół, a każdy jego członek musi mieć możliwość wpływania na sposób ich realizacji.

12. **W stałych odstępach czasu zespół zaangażowany w tworzenie oprogramowania powinien analizować możliwości usprawnienia pracy i dostosowywać swoje dalsze działania do wniosków płynących z tej analizy.** Zwinny zespół stale udoskonala swoją organizację, reguły, konwencje, relacje itp. Członkowie takiego zespołu doskonale zdają sobie sprawę z tego, że środowisko, w którym pracują, stale się zmienia, i wiedzą, iż ich zwinność zależy w dużej mierze od zdolności dostosowywania się do tych modyfikacji.

## Konkluzja

Celem zawodowym każdego programisty i każdego zespołu programistów jest generowanie możliwie jak najwyższej jakości kodu dla pracodawców i klientów. Mimo to nadal przerażający odsetek tego rodzaju projektów kończy się niepowodzeniem lub nie

przynosi korzyści właściwym organizacjom. Środowisko programistów wpadło w pułapkę nieustannego rozwijania procesów, które mimo dobrych intencji dodatkowo utrudniły realizację projektów. Reguły i praktyki zwinnego wytwarzania oprogramowania powstały właśnie po to, by ułatwić zespołom wychodzenie ze spirali rozbudowywania procesów i koncentrowanie się wyłącznie na technikach prowadzących do osiągnięcia właściwych celów.

Kiedy pisano tę książkę, istniało wiele zwinnych procesów i metodyk, spośród których mogliśmy wybierać: SCRUM<sup>4</sup>, Crystal<sup>5</sup>, FDD (od ang. *Feature-Driven Development*)<sup>6</sup>, ADP (od ang. *Adaptive Software Development*)<sup>7</sup> oraz programowanie ekstremalne (ang. *eXtreme Programming — XP*)<sup>8</sup>. Z drugiej strony, zdecydowana większość skutecznych zespołów zwinnych zdołała wybrać takie kombinacje części spośród wymienionych procesów, które tworzą ich własne, autorskie formy zwinności, lekkości. Do najczęściej stosowanej kombinacji należy połączenie metodyk SCRUM i XP, w którym metodykę SCRUM wykorzystuje się do zarządzania wieloma zespołami stosującymi metodykę XP.

## Bibliografia

- [Beck, 99] Kent Beck, *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.
- [Highsmith, 2000] James A., *Highsmith, Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*, Dorset House, 2000.
- [Newkirk, 2001] James Newkirk i Robert C. Martin, *Extreme Programming in Practice*, Addison-Wesley, 2001.

---

<sup>4</sup> Zob. [www.controlchaos.com](http://www.controlchaos.com).

<sup>5</sup> Zob. [http://alistair.cockburn.us/index.php/Crystal\\_methodologies\\_main\\_foyer](http://alistair.cockburn.us/index.php/Crystal_methodologies_main_foyer).

<sup>6</sup> Peter Coad, Eric Lefebvre i Jeff De Luca, *Java Modeling in Color with UML: Enterprise Components and Process*, Prentice Hall, 1999.

<sup>7</sup> [Highsmith, 2000].

<sup>8</sup> [Beck, 99], [Newkirk, 2001].



# Skorowidz

- .NET, 27, 376
- {abstract}, 320
- «C API», 319
- «create», 325
- «delegate», 325, 326
- «function», 319
- «interface», 318
- «local», 325
- «marker», 675
- «parameter», 325
- «persistence», 319
- «structure», 319
- «utility», 319

## A

- A/I, 526
- Abort, 282
- abstract, 319
- Abstract Server, 596, 597
- AbstractDbGatewayTest, 643
- abstrakcja, 175, 184, 189
- abstrakcyjność, 552
- abstrakcyjny interfejs, 532
- abstrakcyjny serwer, 597
- activation, 245, 292
- Active Object, 373, 380
- ActiveObjectEngine, 380, 382
- Acyclic Dependencies Principle, 512
- Acyclic Visitor, 185, 654, 656
  - macierz rzadka, 660
- acykliczny graf skierowany, 514
- adaptacja obiektów, 262
- Adapter, 598, 603
  - forma klasowa, 599
  - forma obiektowa, 599
  - problem modemu, 599
  - zasada LSP, 599
- Adaptive Software Development, 47, 64
- AddItem, 610
- AddItemTransaction, 611
- AddSalariedTransaction, 451
- ADO.NET, 611, 632
- ADP, 47, 64, 511, 512
- afferent couplings, 521
- aggregation, 321
- Agile Alliance, 38
- agility, 149
- agregacja, 321
- akcje, 247
- aktor, 289
- aktualizacja stanu obiektu, 571
- aktywacja, 245, 292
- algorytm sita Eratostenesa, 87
- analiza obiektowa, 193
- analiza podziału na pakiety, 541
- analiza przypadków użycia, 433
- API, 81, 631
- aplikacje internetowe, 756
- Application Programming Interface, 81
- ApplicationRunner, 396, 398
- architektura
  - oprogramowanie, 81
  - wielowarstwowa, 756
  - WWW, 757
- argumenty komunikatów, 292
- ArrayList, 610
- asocjacje, 243, 313
  - klasy, 326
  - kwalifikatory, 327
  - stereotypy, 324
- ASP.NET, 761
- assemblies, 506
- Assembly, 662, 681
- associations, 243
- asynchronous messages, 303
- ATM, 230
- Automated Teller Machine, 230



**B**

BadPartExtension, 685  
 bankomat, 231  
 baza danych, 433, 631, 715, 716  
   odczytywanie danych, 738  
   technologia, 716  
   transakcje, 729  
   wzorce projektowe, 646  
 behavior-driven development, 770  
 biblioteki DLL, 506, 509, 542  
 bill of materials, 660  
 biznesowe decyzje, 66  
 black box tests, 79  
 błędna kompozycja, 323  
 BOM, 660  
 BOMReportTest, 665  
 BomXmlTest, 676  
 boskie klasy, 337  
 bottom-up, 507  
 boundary diagrams, 288  
 bounded, 200  
 brak liczności, 566  
 brak usterek, 160  
 brama do tabeli z danymi, 405  
 Bridge, 604  
 BubbleSorter, 392, 399  
 budowa bazy danych, 716  
 budowa zespołu, 40  
 budowanie  
   model oprogramowania, 250  
   oprogramowanie, 149  
 budżet, 68  
 burn-down charts, 70

**C**

C#, 27  
 C++, 28  
 Ca, 521  
 CASE, 40, 265, 820  
 CCP, 510, 544, 669  
 Ce, 521  
 ciąg główny, 528  
 ciągła integracja, 55  
 ciągłość procesu projektowania, 149  
 Circle, 181

class diagram, 243  
 Classifications, 543, 544, 549, 556  
 ClassificationTransactions, 553, 556  
 Clear, 294  
 Clock, 568  
 ClockDriver, 570, 572, 573, 574  
 ClockDriverTest, 571, 576, 578  
 ClockObserver, 574, 576  
 CLR, 412  
 cohesion, 168  
 CoinInLockedState, 691  
 CoinInUnlockedState, 691  
 collaboration diagram, 246  
 COM, 631  
 Command, 373, 381  
   czasowy podział kodu, 379  
   fizyczny podział kodu, 378  
   transakcje, 377  
   Undo, 379  
 Common Closure Principle, 669  
 Common Language Runtime, 412  
 Common Reuse Principle, 509  
 Composite, 326, 563, 564  
   brak liczności, 566  
   liczność, 566  
   polecenia kompozytowe, 565  
 CompositeCommand, 566  
 composition, 322  
 Computer-Aided Software Engineering, 40  
 ConfigureForUnix, 650  
 Console.In, 389  
 Copy, 159  
   KeyboardReader, 159  
   modyfikacje, 161  
   PrinterWriter, 159  
   projekt zwinny, 163  
   schemat struktury programu, 159  
   zmieniające się wymagania, 161  
 CORBA, 631  
 cotygodniowe kompilacje, 512  
 crossed wires, 339  
 CRP, 509  
 Crystal, 47, 64  
 CSV, 675, 685  
 CsvAssemblyExtension, 684  
 CsvPartExtension, 683  
 CsvPiecePartExtension, 683

cykl w grafie zależności pomiędzy  
komponentami, 515  
czasowy podział kodu, 379  
czytelność kodu, 93

## D

DAG, 514, 516  
DAO, 634  
Data Access Object, 172, 634  
data tokens, 245  
Data Transfer Object, 639  
DatabaseFacade, 647  
DatabaseWriterExtension, 646  
data-driven approach, 185  
Db, 404, 617  
DBC, 198  
DbOrderGateway, 635, 643  
DbOrderGatewayTest, 645  
DbProductGateway, 638, 643  
DbProductGatewayTest, 643  
Decorator, 326, 647, 668  
decyzje  
    biznesowe, 482  
    projektowe, 482  
DedicatedModem, 601, 605  
DedUsers, 600  
definiowanie  
    kontrakty w formie testów  
        jednostkowych, 199  
    odpowiedzialność, 170  
    warunki zakończenia projektu, 67  
degradacja projektu, 158, 159  
dekorator, 326, 647  
DelayedTyper, 384  
delegacja, 227, 387, 398  
delegowanie obserwatora, 584  
depend on abstractions, 215  
Dependency Inversion Principle, 165, 344  
Design By Contract, 198  
diagram skończonych maszyn stanów, 274  
diagramy  
    Boocha, 820  
    dynamiczne, 241  
    fizyczne, 241  
    graniczne, 288  
    implementacji, 240

    konceptyjne, 239, 240  
    niskopoziomowe, 298  
    specyfikacji, 240  
    statyczne, 241  
    STD, 275  
    Warner-Orr, 820  
    współpracy, 246, 259  
    wysokopoziomowe, 298  
    zmiany stanów, 270  
    związki encji, 267  
diagramy klas, 243, 260, 282, 311, 315  
    {abstract}, 320  
    «utility», 319  
    agregacja, 321  
    asocjacje, 313  
    groty strzałek, 314  
    gwiazdka, 313  
    klasy, 312  
    klasy abstrakcyjne, 318, 319  
    klasy asocjacji, 326  
    klasy usługowe, 319  
    klasy zagnieżdżone, 326  
    kompozycja, 322  
    kwalifikatory asocjacji, 327  
    liczność, 323  
    nieprawidłowe cykle agregacji, 322  
    relacje dziedziczenia, 314  
    stereotypy asocjacji, 324  
    stereotypy klas, 318  
    właściwości, 320  
diagramy obiektów, 244, 277  
    obiekty aktywne, 279, 282  
diagramy sekwencji, 245, 291, 293  
    aktywacje, 292  
    argumenty komunikatów, 292  
    komunikaty, 292  
    komunikaty asynchroniczne, 302, 303  
    komunikaty synchroniczne, 302  
    komunikaty wymagające czasu  
        na przesyłanie, 300  
    linie życia, 292  
    niszczenie obiektów, 293  
    obiekty, 292  
    obiekty aktywne, 308  
    pętle, 295, 298  
    scenariusze, 295  
    strzałki ukośne, 300

- diagramy sekwencji
    - tokeny danych, 292
    - tworzenie obiektów, 293
    - warunki, 298
    - warunki wyścigu, 301
    - wątki, 307
    - wiele wątków, 307
    - wrażenia warunkowe, 300
    - wysyłanie komunikatów do interfejsów, 309
  - diagramy stanów, 246, 269
    - hierarchiczne wywoływanie akcji wejścia i wyjścia, 274
    - nadstan, 271
    - przejścia, 271
    - przejście zwrotne, 272
    - przykrywanie przejść wychodzących z nadstanów, 273
    - pseudostan końcowy, 274
    - pseudostan początkowy, 271, 274
    - skończona maszyna stanów, 274
    - stany, 270
    - superstan, 271, 272
    - zdarzenia specjalne, 271
  - diagramy UML, 56, 76, 154, 240
    - CASE, 265
    - dokumentacja, 266
    - dokumentacja wewnętrzna, 255
    - efektywne stosowanie, 251
    - ewolucja diagramów, 262
    - groty strzałek, 314
    - iteracyjne udoskonalanie, 257
    - kod źródłowy, 262
    - komunikacja z innymi programistami, 252
    - mapy drogowe, 253
    - nadużywanie, 249
    - rysowanie, 264, 265
    - tworzenie, 264
    - weryfikacja struktury, 259
    - wyobrażenie o kodzie, 261
    - zachowania, 257
  - dial tone, 301
  - DialModem, 605
  - DialModemController, 605
  - DigitalClock, 570, 589
  - DIP, 165, 215, 344, 388, 401, 516, 540, 591
  - Directed Acyclic Graph, 514
  - DLL, 506, 509
  - dodanie elementu do modelu relacyjnego, 611
  - dokumentacja, 40, 266, 813, 821
    - wewnętrzna, 255
    - wymagania, 80
    - zewnętrzna, 821
  - doskonałość techniczna, 45
  - dostatecznie głęboka analiza, 607
  - dostosowanie do interfejsu, 598
  - DoubleBubbleSorter, 395
  - DTO, 639
  - dual dispatch, 650
  - dynamic diagrams, 241
  - dynamiczna kontrola typów, 536
  - działające oprogramowanie, 40, 44
  - dziedziczenie, 205, 241, 314, 387, 412
  - dzielenie opowieści użytkownika, 65
- ## E
- efferent couplings, 521
  - EJB, 631
  - ekspres do kawy Mark IV Special, 330
    - boskie klasy, 337
    - brakujące metody, 334
    - CoffeeMaker, 358
    - CoffeeMakerAPI, 344
    - ContainmentVessel, 355
    - HotWaterSource, 347, 353
    - implementacja modelu abstrakcyjnego, 343
    - implementacja projektu obiektowego, 366
    - interfejs użytkownika, 339
    - IsReady, 346
    - klasy ulotne, 335
    - komponenty, 359
    - M4ContainmentVessel, 348, 356
    - M4HotWaterSource, 354
    - M4UserInterface, 344, 352
    - M4UserInterface.CheckButton, 348
    - nieprzemysłana abstrakcja, 336
    - niewłaściwe rozwiązanie, 333
    - Pollable, 349, 358
    - przypadek użycia — niegotowe naczynie na kawę, 341
    - przypadek użycia — użytkownik naciska przycisk Brew, 344

przypadek użycia — użytkownik naciska  
   przycisk uruchamiający parzenie, 339  
 przypadek użycia — zakończony proces  
   parzenia kawy, 341  
 przypadek użycia — zużycie całej kawy, 342  
 rozwiązanie, 337  
 skrzyżowane przewody, 339  
 specyfikacja, 330  
 Start, 347  
 TestCoffeeMaker, 360  
 urządzenia, 331  
 UserInterface, 339, 351  
 źródła projektu, 359  
 eliminacja  
   cykl zależności, 513  
   powiązania, 78  
   zależności, 171  
 EmployeeFactory, 537  
 Entity-Relationship, 267  
 ER, 267  
 ErnieModem, 652, 657  
 ErnieModemVisitor, 656  
 EventHandler, 407  
 ewolucja  
   diagramy, 262  
   kod w kierunku wzorca, 568  
 ExplodedCostVisitor, 663  
 Extension Object, 646, 674, 675  
 eXtreme Programming, 47

## F

fabryka, 532  
   inicjalizacja, 558  
   wymienna, 537  
 Facade, 172, 403, 404  
   baza danych, 647  
   stosowanie, 404  
 Factory, 531, 782  
   efekt zastosowania, 533  
   fabryki wymienne, 537  
   kontrola typów, 536  
   podrabianie, 539  
   problem zależności, 534  
   stosowanie, 540  
   testowanie, 538

fasada, 172, 404, 647  
 FDD, 47, 64  
 Feature-Driven Development, 47, 64  
 Finite State Machines, 269  
 FitNesse, 81  
 fizyczny podział kodu, 378  
 fragility, 150  
 FSM, 269, 270, 417, 688  
 FSMError, 707  
 Furnace, 219

## G

garbage collector, 294  
 generalizacja, 314  
 GeneratePrimes, 86  
 generator raportów, 661  
 generowanie  
   kod XML, 675  
   liczby pierwsze, 84  
   raporty, 660  
 GetExtension, 675  
 głęboka kopia, 324  
 gra planistyczna, 57  
 gra w kręgle, 100  
   diagram UML tablicy wyników, 102  
   reguły, 147  
 graf skierowany, 513  
 graf zależności pomiędzy komponentami, 511  
 graficzny interfejs użytkownika, 55, 67, 177  
 GraphicalApplication, 169  
 groty strzałek, 314  
 guards, 245  
 GUI, 55, 67, 169, 177, 709

## H

harmonijna współpraca, 39  
 HAS-A, 314  
 Hashtable, 637  
 HayesModem, 652, 657, 671  
 HayesModemVisitor, 657  
 hermetyzacja, 549  
 heurystyki, 208, 329  
 historie użytkownika, 51

**I**

IDE, 39  
 idealna konfiguracja komponentów, 523  
 identyfikacja abstrakcji, 443  
 if-else, 179  
 imitowanie, 538  
 immobility, 150  
 implementacja
 

- model abstrakcyjny, 343
- projekt obiektowy, 366

 index card, 51  
 inicjalizacja fabryk, 558  
 initial pseudostate, 271  
 InitializeArrayOfBooleans, 93  
 InitializeArrayOfIntegers, 89, 93  
 InitializeSieve, 89  
 InMemoryOrderGateway, 637  
 InMemoryPayrollDatabase, 719  
 InMemoryProductGateway, 639  
 instability, 521  
 Instance, 410  
 IntBubbleSorter, 395  
 intentional programming, 75  
 interaktywne środowiska wytwarzania, 39  
 interface pollution, 225  
 Interface Segregation Principle, 223  
 interfejs użytkownika, 755, 757, 758
 

- bankomat, 230

 interfejsy, 404
 

- API, 81, 631
- klasy, 227
- obiekty, 227
- WWW, 756
- znaczący, 675

 interpretacja tabeli przejść, 694  
 IntSortHandler, 400  
 inversion, 212  
 IS-A, 193, 197, 314  
 ISP, 223, 592  
 Item, 610  
 iteracje, 66, 69  
 iteracyjne udoskonalanie, 257  
 izolacja
 

- klasy, 261
- między akcjami a logiką skończonej maszyny stanów, 699
- testy, 76

**J**

Java, 27  
 jawne zamykanie oprogramowania dla zmian, 184  
 jeden-do-wielu, 313, 409, 566  
 jednostka binarna, 506  
 jedność, 168  
 JEST, 314  
 język
 

- naturalny, 240
- PDL, 820
- UML, 154, 239

 just-in-time requirements, 286

**K**

KeyboardReader, 159  
 kierunek kompilacji, 518  
 kierunek stabilności, 519  
 klasy, 312
 

- abstrakcyjne, 318, 319, 525
- boskie, 337
- dziedziczenie, 314
- stereotypy, 318
- ulotne, 335
- usługowe, 319
- zagnieżdżone, 326

 klasy asocjacji, 313, 326  
 kod XML, 675  
 kod źródłowy, 25, 40, 262, 811  
 kodowanie, 251  
 kompilator maszyny stanów, 276, 700  
 kompletny wykaz elementów funkcjonalności systemu, 64  
 komponenty, 505, 506, 542
 

- błędne zależności, 524
- cykl w grafie zależności, 515
- eliminacja cykli zależności, 513
- graf skierowany, 514
- hermetyzacja, 549
- idealna konfiguracja, 523
- kierunek stabilności, 519
- miary stabilności, 520
- mierzenie abstrakcji, 526
- nieodpowiedzialne, 545
- niezależne, 522
- odległość od ciągu głównego, 528

- odpowiedzialne, 522
- problem spójności, 511
- przerywanie cykli, 516
- stabilność, 511, 519
- stabilność pozycyjna, 520
- strefa wykluczenia, 527
- struktura zależności, 521
- wiązanie, 549
- wskaźnik A, 526
- wskaźnik D, 529
- wysokopoziomowy układ, 524
- zasada acyklicznych zależności, 511
- zasada równoważności wielokrotnego użycia i wydawania, 507
- zasada stabilnych abstrakcji, 525
- zasada stabilnych zależności, 519
- zasada zbiorowego wielokrotnego stosowania, 509
- zasada zbiorowego zamykania, 510
- zasady spójności, 507
- zbyt abstrakcyjne, 528
- zbyt niestabilne, 528
- ziarnistość, 507
- zmienna stabilność, 522
- kompozycja, 322
- kompozyt, 326, 563
- komunikacja z innymi programistami, 252
- komunikaty, 292
  - aktywacja urzędnika, 301
  - argumenty, 292
  - asynchroniczne, 302, 303
  - czas przesyłania, 300
  - rejestrwanie, 309
  - synchroniczne, 302
  - wysyłanie do interfejsów, 309
- konceptyjny diagram UML, 240
- konserwacja aplikacji obiektowych, 236
- konstrukcja, 245
- kontrakty, 199
- kontrola typów, 536
- kontrolki, 776
- koszty pozostałych zadań, 70
- koszyk z zakupami, 610
- krawędzie skierowane, 513
- kształt kodu źródłowego, 261
- kwalifikatory asocjacji, 327

**L**

- lampa biurkowa, 596
- leniwe konstruowanie, 412
- liczby pierwsze, 84
- liczność, 323, 566
- lifelines, 292
- Light, 598
- linie życia, 292
- links, 245, 246
- Liskov Substitution Principle, 189
- Liskov, Barbara, 190
- lista plac, 367
  - AddEmp, 434
  - AddEmployeePresenter, 764
  - AddEmployeePresenterTest, 761
  - AddEmployeeTransaction, 451, 455, 759
  - AddEmployeeView, 769
  - AddEmployeeWindow, 773
  - AddEmployeeWindowTest, 770
  - AddSalariedEmployee, 456
  - AddServiceCharge, 464
  - Affiliation, 446
  - AllInfoIsCollected, 768
  - analiza podziału na pakiety, 541
  - analiza przypadków użycia, 433
  - baza danych, 433, 499, 715
  - baza danych systemu placowego, 452
  - błędny format transakcji, 439
  - ChangeAffiliationTransaction, 476, 478
  - ChangeClassificationTransaction, 471, 473
  - ChangeDirectTransaction, 475
  - ChangeEmployeeTransaction, 467, 470
  - ChangeHoldTransaction, 475
  - ChangeHourlyTransaction, 474
  - ChangeMailTransaction, 475
  - ChangeMemberTransaction, 476, 477, 479
  - ChangeMethodTransaction, 475
  - ChangeNameTransaction, 466, 468, 470
  - ChangeSalariedTransaction, 472
  - ChangeUnaffiliatedTransaction, 476, 479
  - ChangeUnionMember, 477
  - ChgEmp, 439
  - Classifications, 544, 556
  - ClassificationTransactions, 553, 556
  - DateUtil, 496
  - decyzje biznesowe, 482

## lista płac

- decyzje projektowe, 482
- DelEmp, 436
- DeleteEmployee, 457
- DeleteEmployeeTransaction, 458
- diagram klas systemu płacowego, 440
- diagram komponentów, 543, 548
- dodawanie danych pracownika, 369, 434, 450
- dodawanie nowych pracowników, 719
- Dzień wypłat nie przypada dzisiaj, 481
- Dzień wypłat przypada dzisiaj, 482
- Employee, 496
- Employee.Payday, 485
- EnablingAddEmployeeButton, 777
- Execute, 484, 492
- fabryka obiektów, 556
- fabryka obiektów transakcji, 557
- formularz transakcji Dodaj pracownika, 758
- formy wypłat, 446
- GUITransactionSource, 499
- harmonogram wypłat, 444
- hermetyzacja, 549
- HourlyClassification, 489, 492
- identyfikacja abstrakcji, 443
- implementacja, 449
- implementacja interfejsu użytkownika, 759
- inicjalizacja fabryk, 558
- InMemoryPayrollDatabase, 719
- interfejs użytkownika, 755, 758
- karty czasu pracy, 458
- kontrolki, 776
- księgowanie wypłat, 482
- LoadEmployeeOperation, 740, 741, 742, 743, 744, 746, 751
- LoadEmployeeOperationTest, 739
- LoadEmployeeOperationTest.Load
  - ↳EmployeeData, 741
- LoadEmployeeOperationTest.Loading
  - ↳HoldMethod, 744
- LoadEmployeeOperationTest.Loading
  - ↳Schedules, 742, 743
- LoadPaymentMethodOperation, 745, 747, 748, 751
- LoadPaymentMethodOperationTest, 745, 746, 750

- LoadPaymentMethodOperationTest.Load
  - ↳MailMethodCommand, 748
- mechanizm naliczania składek na związki zawodowe, 490
- mierniki, 551, 553
- MockAddEmployeeView, 769
- MockPayrollView, 783
- MockViewLoader, 784
- model dynamiczny abstrakcji
  - harmonogramów wypłat, 445
- model dynamiczny programu głównego, 498
- model dynamiczny scenariusza Dzień wypłat nie przypada dzisiaj, 481
- model dynamiczny scenariusza Dzień wypłat przypada dzisiaj, 482
- model dynamiczny scenariusza Przekazywanie wynagrodzenia, 482
- model dynamiczny transakcji
  - ChangeAddressTransaction, 469
- model dynamiczny transakcji
  - ChangeAffiliationTransaction, 476
- model dynamiczny transakcji
  - ChangeClassificationTransaction, 471
- model dynamiczny transakcji
  - ChangeCommissionedTransaction, 472
- model dynamiczny transakcji
  - ChangeDirectTransaction, 475
- model dynamiczny transakcji
  - ChangeEmployeeTransaction, 468
- model dynamiczny transakcji
  - ChangeHoldTransaction, 475
- model dynamiczny transakcji
  - ChangeHourlyTransaction, 471
- model dynamiczny transakcji
  - ChangeMailTransaction, 475
- model dynamiczny transakcji
  - ChangeMemberTransaction, 476
- model dynamiczny transakcji
  - ChangeMethodTransaction, 475
- model dynamiczny transakcji
  - ChangeNameTransaction, 469
- model dynamiczny transakcji
  - ChangeSalariedTransaction, 472
- model dynamiczny transakcji
  - ChangeUnaffiliatedTransaction, 476



- model dynamiczny transakcji
  - DeleteEmployee, 457
- model dynamiczny transakcji dodawania danych pracownika, 454
- model dynamiczny transakcji
  - PaydayTransaction, 481
- model dynamiczny transakcji
  - SalesReceiptTransaction, 463
- model dynamiczny transakcji
  - ServiceChargeTransaction, 464
- model statyczny abstrakcji
  - harmonogramów wypłat, 445
- model statyczny programu głównego, 498
- model statyczny transakcji
  - ChangeEmployeeTransaction, 467
- model statyczny transakcji
  - PaydayTransaction, 481
- model statyczny transakcji
  - SalesReceiptTransaction, 462
- model statyczny transakcji
  - ServiceChargeTransaction, 463
- Model View Presenter, 760
- MonthlySchedule, 485
- NoAffiliation, 465
- odczytywanie danych o pracownikach, 738
- okna, 770
- okno główne, 778
- okresy rozliczeniowe, 490
- Payday, 441
- PaydayTransaction, 481
- PaydayTransaction.Execute, 484, 492
- PaymentClassification, 442
- PaymentClassification.IsInPayPeriod, 493
- PayrollApplication, 498, 499
- PayrollDatabase, 452, 453, 716, 717, 719
- PayrollDomain, 544
- PayrollMain, 791
- PayrollPresenter, 780
- PayrollPresenterTest, 779
- PayrollTest.HourlyUnionMemberServiceCharge, 494
- PayrollView, 783
- PayrollWindow, 787
- PayrollWindowTest, 786
- PaySingleHourlyEmployee, 487
- PaySingleHourlyEmployeeOnWrongDate, 488
- PaySingleHourlyEmployeeTwoTimeCards, 488
- PaySingleSalariedEmployee, 483
- PaySingleSalariedEmployeeOnWrong
  - ↳Date, 483
- podstawowy model, 441
- połączenie z bazą danych, 722
- PresenterValuesAreSet, 777
- program główny, 498
- projekt warstwy widoku systemu placowego, 778
- przebudowa granic spójności, 558
- Przekazywanie wynagrodzenia, 482
- przynależność związkowa, 446
- przypadki użycia, 431
- raporty o sprzedaży, 458
- RemoteTransactionSource, 499
- SalariedClassification, 440
- SalariedUnionMemberDues, 491
- SalesReceipt, 437
- SalesReceiptTransaction, 462
- SaveEmployeeOperation, 737
- ScheduleCode, 724
- schemat bazy danych, 717
- ServiceCharge, 438
- ServiceChargesSpanningMultiplePay
  - ↳Periods, 495
- ServiceChargeTransaction, 463, 465
- składki na związki zawodowe, 458
- specyfikacja systemu, 368
- SqlPayrollDatabase, 721, 724, 726, 727, 728, 729, 731, 733, 737
- SqlPayrollDatabase.AddEmployee, 738
- SqlPayrollDatabaseTest, 720, 721, 722, 725, 726, 727, 731, 732, 736, 738
- SqlPayrollDatabaseTest.ScheduleGets
  - ↳Saved, 723
- struktura komponentów, 542
- struktura pakietów, 559
- Template Method, 454, 455
- TestChangeHourlyTransaction, 472
- TestChangeNameTransaction, 469
- TestPaySingleHourlyEmployeeNoTime
  - ↳Cards, 486
- TestPaySingleHourlyEmployeeWithTime
  - ↳CardsSpanningTwoPayPeriods, 489

## lista płac

TimeCard, 436, 459  
 TimeCardTransaction, 459, 461  
 Transaction, 450, 718  
 TransactionContainer, 790  
 TransactionContainerTest, 789  
 TransactionImplementation, 557  
 transakcje, 450, 729, 732  
 UnionAffiliation, 477, 492  
 UnionAffiliation.CalculateDeductions, 493  
 uproszczona specyfikacja systemu, 368, 432  
 usunięcie danych pracownika, 370, 436, 456  
 ViewLoader, 782, 783  
 wartości mierników, 555  
 WeeklySchedule, 490  
 wiązanie komponentów, 549  
 wielokrotne wykorzystanie kodu, 546  
 WindowViewLoader, 785  
 WindowViewLoaderTest, 784  
 wygenerowanie listy płac na dany dzień,  
 372, 441  
 wynagradzanie pracowników, 480  
 wynagradzanie pracowników  
 etatowych, 483  
 wynagradzanie pracowników  
 zatrudnionych w systemie  
 godzinowym, 486  
 wynagrodzenia wypłacane  
 pracownikom, 444  
 wysłanie informacji o opłacie na rzecz  
 związku zawodowego, 371, 438  
 wysłanie karty czasu pracy, 370, 436  
 wysłanie raportu o sprzedaży, 370, 437  
 wyznaczanie wysokości wynagrodzenia  
 pracownika, 442  
 zarządzanie zmianami, 547  
 zasada otwarty-zamknięty, 441  
 zasada pojedynczej odpowiedzialności, 434  
 zasada zbiorowego zamykania, 544  
 zmiana danych pracowników, 466  
 zmiana formy wynagradzania  
 pracownika, 471  
 zmiana szczegółowych danych  
 pracownika, 371, 439  
 Logger, 309  
 login, 710  
 LoudDialModem, 670, 671, 674  
 LSP, 189, 190, 224, 591

## Ł

łącza, 245, 246

## M

MA, 314  
 macierz rzadka, 660  
 main sequence, 528  
 Manifest zwinnego wytwarzania  
 oprogramowania, 39  
 mapy drogowe, 253  
 marker interface, 675  
 maszyna stanów, 247, 270, 687  
 mechanizm  
 odzyskiwanie pamięci, 294  
 rejestrowanie komunikatów, 309  
 utrwalanie, 172  
 weryfikacja typów w czasie  
 wykonywania, 190  
 wielodziedziczenie, 582  
 Mediator, 403, 405  
 metafora, 59  
 metaphore, 59  
 metody abstrakcyjne, 318  
 metodyka spirali, 820  
 metodyki zwinne, 64  
 miary stabilności, 520  
 middleware, 55  
 mierniki, 551  
 postęp prac nad projektem, 45  
 zarządzanie zależnościami, 507  
 mierzenie abstrakcji, 526  
 migawka, 278  
 Mock Object, 77  
 mocking, 538  
 MockTimeSink, 573, 576, 577, 586  
 MockTimeSource, 575, 579, 583, 586, 588  
 model, 249  
 model obiektów koszyka z zakupami, 610  
 model oprogramowania, 250  
 model specyfikacji, 241  
 Model View Presenter, 755, 760  
 model wpychania, 585  
 model wyciągania, 590  
 modelowanie  
 oprogramowanie, 249  
 zachowania, 257

modem, 599  
 Modem, 651, 670  
 ModemConnectionController, 605  
 ModemDecorator, 673  
 ModemDecoratorTest, 672  
 ModemImplementation, 171  
 ModemVisitor, 656  
 ModemVisitorTest, 653, 659  
 moduły  
     niskopoziomowe, 212  
     wysokopoziomowe, 212  
 modyfikacje programu, 161  
 Monostate, 409, 415, 417  
     dziedziczność, 417  
     efektywność, 417  
     ograniczony zasięg, 417  
     polimorfizm, 417  
     przejrzystość, 417  
     wady, 417  
     zajmowane miejsce, 417  
     zalety, 417  
 most, 605  
 motywacja programistów, 45

## N

nadmierna złożoność, 150, 156  
 nadstan, 271  
 nadużywanie diagramów UML, 249  
 name-only dependency, 534  
 narzędzia  
     CASE, 265  
     projektowanie obiektowe, 206  
 narzucanie strategii, 405  
 następny poranek, 518  
 naturalna struktura, 181  
 needless complexity, 150  
 needless repetition, 150  
 negocjacje zasad kontraktu, 41  
 NHibernate, 635  
 niedostosowanie do rzeczywistości, 150, 156  
 nieelastyczność, 150, 156  
 niepotrzebne powtórzenia, 150, 157  
 nieprawidłowe cykle agregacji, 322  
 nieprzejrzystość, 150, 157  
 niestabilność, 521, 552  
 niszczenie obiektów, 293

notacja  
     Boocha, 320  
     projekt oprogramowania, 820  
 null, 426  
 Null Object, 425, 426  
 NUnit, 776  
 NUnitForms, 776

## O

obiekt dostępu do danych, 172  
 obiekt rozszerzenia, 646  
 obiekt zastępczy, 77  
 obiektowy język programowania, 175  
 obiekty, 292  
     aktywne, 279, 282, 308  
     niszczenie, 293  
     tworzenie, 293  
 obiekty transferu danych, 639  
 object diagram, 244  
 Object-Oriented Design, 54  
 Object-Oriented Programming Language, 175  
 ObservableClock, 582  
 Observer, 567, 589  
     delegowanie, 584  
     model wyciągania, 590  
     modele, 590  
     zasada otwarty-zamknięty, 591  
 obszar odpowiedzialności, 169  
 OCP, 164, 174, 441, 443, 444, 510, 591  
 oczekiwania klienta, 43  
 oddzielanie  
     reguły biznesowe od szczegółowej  
     implementacji, 631  
     wzajemnie powiązane  
     odpowiedzialności, 171  
 odległość od ciągu głównego, 552  
 odwracanie relacji własności, 213  
 odwracanie zależności, 165, 211  
     zależności łączące aplikację z warstwą  
     izolującą, 633  
 odzyskiwanie pamięci, 294  
 off-hook message, 301  
 ogólność, 552  
 ograniczanie liczby obiektów wybranych  
     klas, 415  
 okablowanie, 376

okno, 770  
 OOD, 54, 206  
 OOPL, 175  
 opacity, 150  
 open workspace, 56  
 Open/Closed Principle, 164, 174  
 opowieści użytkownika, 50, 65  
   dzielenie, 65  
   scalanie, 65  
 oprogramowanie, 40, 811  
   pośredniczące, 55, 631  
   wolne od wzajemnych związków, 74  
 ORB, 58  
 OrderData, 623  
 OrderGateway, 634, 635  
 OrderProxy, 626  
 organizacja wysokopoziomowa, 505  
 otwarta przestrzeń pracy, 56  
 otwarte-zamknięte, 164, 174

## P

pakiety, 505, 506  
 pakowanie systemu płacowego, 503  
 Part, 661, 679  
 PartCountVisitor, 664  
 PartExtension, 680  
 PartVisitor, 663  
 PassInLockedState, 691  
 PassInUnlockedState, 691  
 Payroll, 538  
 PayrollApplication, 542  
 PayrollDatabase, 716, 719  
 PayrollDomain, 544  
 PDL, 820  
 pętle, 295, 298  
   wyrażenia warunkowe, 300  
 physical diagrams, 241  
 PiecePart, 662, 680  
 Pierwsze prawo Martina dotyczące  
   dokumentacji, 41  
 pisanie  
   przypadki użycia, 286  
   testy, 76  
 plan iteracji, 51  
 plan wydania, 52  
 planning game, 57

planowanie, 63  
   iteracje, 66  
   modyfikacje, 161  
   wydania, 66  
   zadania, 67  
 podrabianie, 538  
   fabryka, 539  
 podwójny przydział, 650, 654  
 podział na pakiety, 541  
 podział na warstwy, 212  
 pojedyncza odpowiedzialność, 167  
 polecenia kompozytowe, 565  
 polecenie, 373, 374  
 polimorfizm, 177, 189, 417  
 polityka wysokiego poziomu, 217  
 połączenie z bazą danych, 722  
 połowa opowieści użytkownika, 68  
 positional stability, 520  
 postrzeganie kodu jako projektu  
   oprogramowania, 813  
 pośrednik, 172, 326, 610  
 praktyki programowania ekstremalnego, 50  
 prawidłowość modelu, 197  
 primary course, 286  
 PrimeGenerator, 87  
 PrinterWriter, 159  
 priorytety operatorów, 111  
 problemy  
   modem, 599  
   spójność komponentów, 511  
   zależności, 534  
 proces  
   wytwarzanie oprogramowania, 816  
   zwinny, 45  
 ProductGateway, 638  
 ProductProxy, 619, 621, 622  
 program, 334  
 programiści, 39  
 programming by coincidence, 115  
 programowanie  
   intencjonalne, 75  
   obiektowe, 812  
   przez przypadkową zbieżność, 115  
   w parach, 53  
   zwinne, 37, 99, 165

- programowanie ekstremalne, 25, 47, 49
    - ciągła integracja, 55
    - gra planistyczna, 57
    - krótkie cykle, 51
    - metafora, 59
    - moduły, 54
    - opowieści użytkownika, 50
    - otwarta przestrzeń pracy, 56
    - plan iteracji, 51
    - plan wydania, 52
    - praktyczne rozwiązania, 58
    - programowanie w parach, 53
    - prosty projekt, 57
    - refaktoryzacja, 54, 59
    - równe tempo, 56
    - system, 60
    - tempo, 56
    - testy akceptacyjne, 52
    - wspólna własność, 54
    - wytwarzanie sterowane testami, 54
    - zadania, 52
    - zasady, 58
    - zespół, 50
  - projektowanie, 24, 134
    - obiektywne, 54, 812
    - oprogramowanie, 814
    - przez kontrakt, 198
    - wstępujące, 518
    - zstępujące, 518
    - zwinne, 149, 153
  - projekty, 38, 814
    - początkowe, 159
    - poprzedzone testami, 75, 486
    - XP, 50
  - Proxy, 172, 326, 610, 614
    - implementacja wzorca, 615
    - model dynamiczny, 615
    - model statyczny, 614
  - przebieg główny, 286
  - przebiegi alternatywne, 287
  - przebudowa granic spójności, 558
  - przejścia, 271
    - między stanami, 247
    - zwrotne, 272
  - przekazywanie informacji, 45
  - przenośność między platformami, 412
  - zapełnienie sterty, 200
  - przerywanie cykli zależności łączących komponenty, 516
  - przetwarzanie rozproszone, 712
  - przewidywanie zmian, 181
  - przyczyna zmiany, 170
  - przygotowywanie punktów zaczepienia, 182
  - przypadki testowe, 111
  - przypadki użycia, 285
    - pisanie, 286
    - przebieg główny, 286
    - przebiegi alternatywne, 287
    - relacje, 289
    - widoczne zdarzenia, 286
    - zdarzenia, 286
  - pseudostany
    - końcowy, 274
    - początkowy, 271, 274
  - pull model, 590
  - push model, 585
  - PutUncrossedIntegersIntoResult, 92
- ## Q
- QuickBubbleSorter, 401
  - QuickEntryMediator, 405, 407
- ## R
- race condition, 301
  - RDBMS, 500
  - reagowanie na zmiany, 42
  - realizacja, 315
  - realizacja planu, 42
  - realizes relationship, 315
  - refactoring, 54, 59
  - refaktoryzacja, 54, 59, 83, 87
    - ReSharper, 87
  - reguły gry w kręgle, 147
  - rejestrwanie komunikatów, 309
  - relacje
    - dziedziczenie, 314
    - IS-A, 193, 197, 209
    - jeden-do-jednego, 566
    - kompozycja, 322
    - realizacja, 315
  - relacyjny model danych koszyka, 611
  - relacyjny system zarządzania bazą danych, 500

relational cohesion, 551  
 release plan, 52  
 REP, 507, 508, 546  
 ReSharper, 87  
 Reuse/ Release Equivalence Principle, 508  
 rigidity, 150  
 rozbudowany projekt, 251  
 RTC, 384  
 Run-To-Completion, 384  
 rysowanie diagramów, 264, 265

## S

SalesReceipt, 550  
 SalesReceiptTransaction, 550  
 SAP, 525  
 satyra na dwa przedsiębiorstwa, 793  
 skalanie opowieści użytkownika, 65  
 scenariusze, 295  
 SCRUM, 47, 64  
 SDP, 519, 522, 523  
 segregacja interfejsów, 223  
 Self-Shunt, 538  
 Sensor, 565  
 separacja
 

- przez delegację, 227
- przez wielokrotne dziedziczenie, 229

 sequence diagram, 245  
 Set, 201  
 Shape, 177  
 ShapeFactory, 533, 534  
 silna analiza, 793  
 Single-Responsibility Principle, 168  
 Singleton, 409, 410
 

- efektywność, 413
- implementacja wzorca, 412
- leniwe konstruowanie, 412
- możliwość stosowania dla dowolnej klasy, 412
- niedziedziczność, 413
- nieprzejrzystość, 413
- niezdefiniowany proces destrukcji, 412
- przypadki testowe, 411
- tworzenie przez dziedziczenie, 412
- wady, 412
- zalety, 412

 sito Eratostenesa, 87  
 skończona maszyna stanów, 269, 417, 687
 

- interpretacja tabeli przejść, 694
- izolacja między akcjami a logiką, 699
- przejścia między stanami, 693
- State, 696
- tabela przejść, 693
- testowanie akcji, 692
- wewnętrzny zasięg zmiennej stanu, 691
- zagnieżdżone wyrażenia switch-case, 688
- zastosowanie, 709

 skrzyżowane przewody, 339  
 Sleep, 282  
 SleepCommand, 382  
 SMC, 276, 700  
 SocketServer, 282  
 SomeApp, 533  
 SortHandler, 399  
 sortowanie bąbelkowe, 392  
 spare, 147  
 sparse matrix, 660  
 specyfikacja zachowań, 770  
 spełnianie oczekiwań klienta, 43  
 spiral development, 820  
 spoofing, 538  
 spójność komponentów, 507  
 spójność relacyjna, 551  
 SQL Server, 716, 732  
 SqlTransaction, 732  
 SRP, 168, 434, 444, 510, 669  
 stabilność, 511, 519
 

- pozycyjna, 520

 Stable-Abstractions Principle, 525  
 stalwart-analysis, 793  
 stany, 270  
 Start, 282  
 State, 687, 696, 699
 

- skończona maszyna stanów, 688
- Turnstile, 696, 698
- wady, 699
- wewnętrzny zasięg zmiennej stanu, 691
- zalety, 699

 State Machine Compiler, 276, 700  
 State Transition Diagram, 270  
 State Transition Tables, 275  
 static diagrams, 241

statyczna kontrola typów, 536  
 STD, 270, 275  
 stereotypy
 

- asocjacje, 324
- klasy, 318

 sterowanie
 

- interakcja z interfejsem GUI, 711
- polecenie, 375
- przez dane, 185

 sterta, 200  
 stosowanie mierników, 553  
 strategia
 

- brak usterek, 160
- wysoki poziom, 217

 Strategy, 176, 387, 396, 699  
 strażnik, 245  
 strefy
 

- bezużyteczność, 528
- ból, 527
- wykluczenie, 527

 strike, 147  
 string, 532  
 struktura komponentów, 542  
 struktura zależności, 521  
 struktury
 

- nieograniczone, 200
- ograniczone, 200

 STT, 275  
 stymulowanie zmian, 183  
 superstan, 271, 272  
 superstate, 271  
 Switch, 597  
 Switchable, 597  
 switch-case, 179, 688, 692  
 sygnał wybierania, 301  
 symptomy złego projektu, 149, 154
 

- nadmierna złożoność, 156
- niedostosowanie do rzeczywistości, 156
- nieelastyczność, 156
- niepotrzebne powtórzenia, 157
- nieprzejrzystość, 157
- szytywność, 155
- wrażliwość, 155

 symulowanie
 

- stan połączenia, 603
- zdarzenia, 538

synchronous messages, 302  
 syndrom następnego poranka, 518  
 system, 60  
 system listy płac, 368  
 System.Data, 404  
 szacowanie czasu i kosztu implementacji, 65  
 sztywność, 150, 155  
 szybkość implementacji opowieści
 

- użytkownika, 65

## Ś

śledzenie postępu, 69

## T

tabela przejść, 693
 

- przejścia między stanami, 275

 Table Data Gateway, 405, 634
 

- test bram DB, 643
- testowanie w pamięci, 642

 TDD, 27, 54, 536  
 TDG, 634  
 technika
 

- podwójny przydział, 654
- sterowanie przez dane, 185

 technologia
 

- .NET, 28
- bazy danych, 716

 Template Method, 176, 387, 388, 454, 669
 

- błędne zastosowanie, 392

 test bram DB, 643  
 Test-Driven Development, 27, 54  
 test-first design, 486  
 testowanie, 73
 

- akcje, 692
- eliminowanie powiązań, 78
- izolacja testów, 76
- konstrukcje TDG w pamięci, 642
- testy akceptacyjne, 79
- testy jednostkowe, 79

 TestSleepCommand, 381  
 testy
 

- biała skrzynka, 79
- czarna skrzynka, 79
- jednostkowe, 73, 79, 86, 538

testy akceptacyjne, 52, 79, 80, 81  
 architektura oprogramowania, 81  
 TextReader, 389  
 TextWriter, 389  
 Thread, 282  
 TimeCard, 550  
 TimeCardTransaction, 550  
 TimeSink, 572, 576  
 TimeSource, 571, 572, 575, 579  
 tokeny danych, 245, 292  
 ToXML, 674  
 Transaction, 450  
 TransactionImplementation, 557  
 transakcje, 377, 450, 729, 732  
 transitions, 271  
 TreeMap, 244  
 trwałość, 171  
 try-catch, 426  
 Turnstile, 420, 688, 694, 696, 698, 701, 703  
 TurnstileActions, 701  
 TurnstileController, 689, 692  
 TurnstileFSM, 702  
 TurnstileState, 697  
 TurnstileTest, 418, 690  
 tworzenie  
 diagramy UML, 264  
 diagramy współpracy, 259  
 instancje klas, 532  
 mapy drogowe, 253  
 obiekty, 293

## U

UML, 56, 154, 239  
 diagramy klas, 243  
 diagramy obiektów, 244  
 diagramy sekwencji, 245  
 diagramy stanów, 246  
 diagramy współpracy, 246  
 pakiety, 506  
 unbounded, 200  
 UnboundedSet, 200  
 Undo, 379  
 Unified Modeling Language, 56, 239  
 uniwersalność, 373  
 UNIX, 650  
 UnixModemConfigurator, 652, 658

User, 413  
 user story, 51  
 UserDatabase, 413, 414  
 usuwanie związków między komponentami, 78  
 utility classes, 319  
 utrzymywanie  
 ciągłość wytwarzania, 45  
 projekt w dobrym stanie, 165  
 uwierzytelniony dostęp, 413

## V

vapor classes, 335  
 velocity graph, 69  
 ViewLoader, 782  
 virtual, 194  
 viscosity, 150  
 Visitor, 647, 649, 650  
 efekt zastosowania, 651  
 generowanie raportów, 660  
 zastosowanie, 660, 667

## W

warstwa izolująca, 632  
 warstwy, 212  
 warunki, 298  
 wyścig, 301  
 zakończenie projektu, 67  
 wątki, 307, 384  
 wejście, 271  
 weryfikacja  
 struktury, 259  
 typy w czasie wykonywania, 190  
 wartości null, 426  
 wewnętrzna struktura systemu, 278  
 wewnętrzny zasięg zmiennej stanu, 691  
 węzły grafu, 513  
 white box tests, 79  
 wiązanie komponentów, 549  
 widoczne zdarzenia, 286  
 wielodziedziczenie, 582  
 wielokrotne dziedziczenie, 229  
 wielokrotne wykorzystanie kodu, 395, 546  
 wielozadaniowość, 373  
 Windows Forms, 757  
 wizytator, 647  
 acykliczny, 185



- wrażliwość, 150, 155
  - wskaźniki
    - A, 526
    - D, 529
    - zarządzanie zależnościami, 530
  - współpraca
    - programiści, 39
    - z klientem, 41
  - wstępne poznawanie wymagań, 64
    - dzielenie opowieści użytkownika, 65
    - scalanie opowieści użytkownika, 65
  - WWW, 756
  - wyjście, 271
  - wykresy
    - A/I, 526
    - szybkości, 69
    - wypalania, 70
  - wykrywanie zależności cyklicznych, 312
  - wymagania, 41, 42
    - dokładnie na czas, 286
    - zmiany, 42, 44, 163
  - wyobrażenie o kodzie, 261
  - wyodrębnianie, 205, 207
  - wrażenia
    - rekurencyjne, 298
    - warunkowe, 300
  - wysokopoziomowa polityka działania
    - graficznych interfejsów użytkownika, 709
  - wysokopoziomowa strategia, 217
  - wysokopoziomowe diagramy, 298
  - wysokopoziomowy układ komponentów, 524
  - wysyłanie komunikatów do interfejsów, 309
  - wytwarzanie sterowane testami, 27, 54, 74, 172, 536
  - wytwarzanie sterowane zachowaniami, 770
  - wytwarzanie zwinne, 35
  - wzorce projektowe
    - Abstract Server, 596
    - Active Object, 380
    - Acyclic Visitor, 185, 654
    - Adapter, 598
    - baza danych, 646
    - Bridge, 604
    - Command, 373
    - Composite, 563
    - Decorator, 647, 668
    - Extension Object, 646, 674
    - Facade, 404, 647
    - Factory, 531, 782
    - Mediator, 405
    - Mock Object, 77
    - Model View Presenter, 755, 760
    - Monostate, 415
    - Null Object, 425
    - Observer, 567, 589
    - Proxy, 610
    - Singleton, 410
    - State, 687, 696
    - Strategy, 176, 396
    - Table Data Gateway, 634
    - Template Method, 176, 388, 669
    - Visitor, 647, 649
  - wzorec testowy Self-Shunt, 538
  - wzrost efektywności, 57
- X**
- XML, 674, 685
  - XmlAssemblyExtension, 683
  - XmlPartExtension, 682
  - XmlPiecePartExtension, 682
  - XP, 25, 47, 50
- Z**
- zachowania, 197, 257
  - zadania, 52, 67
    - RTC, 384
  - zagnieżdżone wyrażenia switch-case, 688
  - zakończenie projektu, 46
  - zależności
    - moduły, 212
    - nazwy, 534
    - od abstrakcji, 215
    - między klasami, 311
    - przychodzące, 521
    - w kodzie źródłowym, 314
    - wychodzące, 521
  - zanieczyszczanie interfejsów, 223, 225
  - zarządzanie
    - projekty, 69
    - struktura zależności pomiędzy komponentami, 513
    - zasady projektowania obiektowego, 591
    - zmiany, 547

- zasada acyklicznych zależności, 511, 512
- zasada cotygodniowych kompilacji, 512
- zasada eliminowania ścisłych związków, 79
- zasada odwracania zależności, 165, 211, 344, 388, 516
  - odkrywanie niezbędnych abstrakcji, 217
  - odwracanie relacji własności, 213
  - podział na warstwy, 212
  - stosowanie, 216
  - zależności od abstrakcji, 215
- zasada otwarte-zamknięte, 164, 173, 174, 441, 444, 451, 510
  - abstrakcja, 175, 184
  - jawne zamykanie oprogramowania dla zmian, 184
  - naruszenia zasady, 177
  - naturalna struktura, 181
  - Observer, 591
  - otwarte na rozszerzenia, 174
  - pełna zgodność z zasadą, 180
  - przewidywanie zmian, 181
  - przygotowywanie punktów zaczepienia, 182
  - punkty zaczepienia, 182
  - sterowanie przez dane, 185
  - Strategy, 176
  - stymulowanie zmian, 183
  - Template Method, 176
  - zamknięte dla modyfikacji, 174
  - zapewnianie zamknięcia, 185
- zasada podstawiania Liskov, 189, 190, 224
  - definiowanie kontraktów, 199
  - heurystyki, 208
  - IS-A, 197
  - konwencje, 208
  - naruszenie zasady, 190, 192
  - Observer, 591
  - prawidłowość modelu, 197
  - projektowanie przez kontrakt, 198
  - rozwiązanie niezgodne z zasadą, 203
  - rozwiązanie zgodne z zasadą, 204
  - wyodrębnianie, 205
- zasada pojedynczej odpowiedzialności, 167, 168, 434, 444, 451, 510, 669
  - definiowanie odpowiedzialności, 170
  - oddzielanie wzajemnie powiązanych odpowiedzialności, 171
  - przyczyna zmiany, 170
  - trwałość, 171
- zasada równoważności wielokrotnego użycia i wydawania, 507, 508, 546
- zasada segregacji interfejsów, 223, 227
  - interfejs użytkownika bankomatu, 230
  - interfejsy klas, 227
  - interfejsy obiektów, 227
  - odrębne interfejsy, 225
  - odrębne klasy klienckie, 225
  - separacja przez delegację, 227
  - separacja przez wielokrotne dziedziczenie, 229
  - zanieczyszczanie interfejsów, 223
- zasada spójności komponentów, 507
- zasada stabilnych abstrakcji, 525
- zasada stabilnych zależności, 519
- zasada zależności od abstrakcji, 215
- zasada zbiorowego wielokrotnego stosowania, 509
- zasada zbiorowego zamykania, 510, 544, 669
- zdarzenia, 247, 286
  - specjalne, 271
- zegar cyfrowy, 568
- zero defects, 160
- zespoły, 40, 46
- zestawienia materiałowe, 660
- zestawy, 506
- ziarnistość, 507
  - ponowne użycie, 507
  - wydania, 507
- złe projekty, 149, 154
- zmiany, 42, 163
  - wymagania, 44, 161
- zmienna stabilność komponentów, 522
- znormalizowana odległość od ciągu głównego, 552
- ZoomModem, 652, 658
- ZoomModemVisitor, 657
- zwalnianie obiektu, 294
- związki
  - jeden-do-wielu, 313
  - przychodzące, 551
  - wychodzące, 552
- zwinnosć, 149

## Z

źródła porażek projektów, 154

# 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**

# Agile Programowanie zwinne

Zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w **C#**

W związku ze stale rosnącymi oczekiwaniami użytkowników oprogramowania produkcja systemów informatycznych wymaga dziś korzystania z usystematyzowanych metod zarządzania. Projekt informatyczny, przy którym nie używa się sensownej metodologii wytwarzania, jest skazany na porażkę – przekroczenie terminu, budżetu i niespełnienie wymagań funkcjonalnych. Kierowanie projektem zgodnie z określonymi zasadami również nie gwarantuje sukcesu, lecz znacznie ułatwia jego osiągnięcie. Na początku 2001 roku grupa ekspertów zawiązała zespół o nazwie Agile Alliance. Efektem prac tego zespołu jest metodologia zwinnego wytwarzania oprogramowania – Agile.

Książka „Agile. Programowanie zwinne: zasady, wzorce i praktyki zwinnego wytwarzania oprogramowania w C#” to podręcznik metodologii Agile przeznaczony dla twórców oprogramowania korzystających z technologii .NET. Dzięki niemu poznasz podstawowe założenia i postulaty twórców Agile i nauczysz się stosować je w praktyce. Dowiesz się, jak szacować terminy i koszty, dzielić proces wytwarzania na iteracje i testować produkt. Zdobędziesz wiedzę na temat refaktoryzacji, diagramów UML, testów jednostkowych i wzorców projektowych. Przeczytasz także o publikowaniu kolejnych wersji oprogramowania.

- ▶ Techniki programowania ekstremalnego
- ▶ Planowanie projektu
- ▶ Testowanie i refaktoryzacja
- ▶ Zasady zwinnego programowania
- ▶ Modelowanie oprogramowania za pomocą diagramów UML
- ▶ Stosowanie wzorców projektowych
- ▶ Projektowanie pakietów i komponentów

**Przekonaj się, ile czasu i pracy zaoszczędzisz, stosując w projektach metodologię Agile.**

**Helion** 

 [helion.pl](http://helion.pl)

 **HELION SA**  
ul. Kościuszki 1c  
44-100 Gliwice  
tel.: 32 230 98 63  
helion@helion.pl

*Sprawdź nasze szkolenia!*

**SZKOLENIA**



**AKADEMIA IT & BUSINESS**

[WWW.SZKOLENIA.HELION.PL](http://WWW.SZKOLENIA.HELION.PL)

**KOD KORZYŚCI**  
Sięgnij po więcej! ▶



ISBN 978-83-283-5567-5



9 788328 355675