



ZMIEN SPOSÓB MYŚLENIA O PROJEKTOWANIU  
SYSTEMÓW INFORMATYCZNYCH!

**ERIC EVANS**

# DOMAIN-DRIVEN DESIGN

Zapanuj nad złożonym  
systemem informatycznym

**Helion** 

Tytuł oryginału: Domain-Driven Design: Tackling Complexity in the Heart of Software

Tłumaczenie: Rafał Szpoton

Projekt okładki: Studio Gravite / Olsztyn

Obarek, Pokoński, Pazdrijowski, Zaprucki

ISBN: 978-83-283-0525-0

Authorized translation from the English language edition, entitled: DOMAIN-DRIVEN DESIGN: TACKLING COMPLEXITY IN THE HEART OF SOFTWARE; ISBN 0321125215; by Eric Evans; published by Pearson Education, Inc, publishing as Addison Wesley.  
Copyright © 2004 by Eric Evans.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc. Polish language edition published by HELION S.A. Copyright © 2015.

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.

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/domdri.zip>

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/domdri>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

---

# SPIS TREŚCI

---

<i>Przedmowa</i> .....	15
<i>Wstęp</i> .....	17
<i>Podziękowania</i> .....	27

## *Część I*

Zastosowanie modelu dziedziny .....	29
-------------------------------------	----

Rozdział 1. <i>Przetwarzanie wiedzy</i> .....	35
---	----

Elementy wydajnego modelowania .....	40
Przetwarzanie wiedzy .....	41
Ciągła nauka .....	44
Projekt bogaty w wiedzę .....	45
Modele dogłębne .....	48

Rozdział 2. <i>Komunikacja i użycie języka</i> .....	51
--	----

Język wszechobecny .....	52
Modelowanie na głos .....	58
Jeden zespół, jeden język .....	60
Dokumenty i diagramy .....	63
<i>Spisane dokumenty projektowe</i> .....	65
<i>Wykonywalna podstawa</i> .....	68
Modele objaśniające .....	68

Rozdział 3. <i>Związananie modelu z implementacją</i> .....	71
---	----

Projektowanie sterowane modelem .....	73
Paradygmaty modelowania i narzędzia wspierające .....	76
<i>Projekt mechaniczny</i> .....	79
<i>Projekt sterowany modelem</i> .....	80
Odkrywanie szkieletu — dlaczego modele są ważne dla użytkowników .....	83
Modelowanie praktyczne .....	86

## Część II

### Elementy składowe projektu

sterowanego modelem ..... 89

Rozdział 4. *Wyizolowanie dziedziny* ..... 93

Architektura warstwowa ..... 94

*Powiązanie warstw* ..... 99

*Szkielety architektury* ..... 100

To w warstwie dziedziny żyje model ..... 101

Antywzorec inteligentnego interfejsu użytkownika ..... 102

Inne rodzaje izolacji ..... 106

Rozdział 5. *Wyrażenie modelu w programie* ..... 107

Asocjacje ..... 109

ENCJE (zwane również obiektami referencyjnymi) ..... 115

*Modelowanie ENCJI* ..... 120

*Projektowanie operacji na tożsamości* ..... 121

WARTOŚCI ..... 125

*Projektowanie OBIEKTÓW WARTOŚCI* ..... 128

*Projektowanie asocjacji korzystających z WARTOŚCI* ..... 131

USŁUGI ..... 132

*USŁUGI a wyizolowana warstwa dziedziny* ..... 134

*Ziarnistość* ..... 136

*Dostęp do USŁUG* ..... 137

MODUŁY (zwane również PAKIETAMI) ..... 138

*MODUŁY zwinne (agile modules)* ..... 140

*Pułapki tworzenia pakietów na podstawie wymogów infrastruktury* ..... 142

Paradygmaty modelowania ..... 146

*Dlaczego dominuje paradygmat obiektowy?* ..... 146

*Nieobiekty w świecie obiektowym* ..... 149

*Utrzymywanie PROJEKTU STEROWANEGO*

*MODELEM w przypadku łączenia paradygmatów* ..... 150

Rozdział 6. *Cykl życia obiektu dziedziny* ..... 153

AGREGATY ..... 155

FABRYKI ..... 166

*Wybór FABRYK oraz ich miejsc* ..... 169

*Kiedy potrzebujesz jedynie konstruktora* ..... 171

*Projektowanie interfejsu* ..... 173

<i>Gdzie mieści się logika niezmienników?</i> .....	174
<i>FABRYKI ENCJI a FABRYKI WARTOŚCI</i> .....	174
<i>Odtwarzanie zachowanych obiektów</i> .....	175
REPOZYTORIA .....	178
<i>Odpytywanie REPOZYTORIUM</i> .....	184
<i>Kod klienta, w przeciwieństwie do programistów,</i> <i>ignoruje implementację REPOZYTORIUM</i> .....	185
<i>Implementacja REPOZYTORIUM</i> .....	186
<i>Praca ze szkieletami architektury</i> .....	188
<i>Relacje z FABRYKAMI</i> .....	189
Projektowanie obiektów dla relacyjnych baz danych .....	190
Rozdział 7. <i>Użycie języka — przykład rozszerzony</i> .....	195
Prezentacja systemu logistycznego dla ładunku .....	195
Izolowanie dziedziny — wprowadzenie aplikacji .....	198
Rozróżnianie ENCJI oraz WARTOŚCI .....	199
<i>Role (rola) oraz inne atrybuty</i> .....	201
Projektowanie asocjacji w dziedzinie logistyki morskiej .....	201
Granice AGREGATU .....	203
Wybór REPOZYTORIÓW .....	204
Przeglądanie scenariuszy .....	206
<i>Przykładowa funkcjonalność aplikacji — zmiana miejsca</i> <i>przeznaczenia ładunku</i> .....	206
<i>Przykładowa funkcjonalność aplikacji — powtórzenie operacji</i> ....	206
Tworzenie obiektów .....	207
<i>FABRYKI oraz konstruktory klasy Cargo</i> .....	207
<i>Dodanie operacji obsługi</i> .....	208
Przerwa na refaktoring — projekt alternatywny AGREGATU Cargo .....	209
MODUŁY w modelu logistyki morskiej .....	213
Nowa funkcjonalność — sprawdzanie przydziału .....	215
<i>Łączenie dwóch systemów</i> .....	216
<i>Wzbogacanie modelu — segmentacja biznesu</i> .....	217
<i>Poprawa wydajności</i> .....	219
Ostateczna wersja .....	220

### Część III

## Refaktoryzacja ku głębszemu zrozumieniu ..... 223

### Rozdział 8. *Moment przełomowy* ..... 229

Historia pewnego przełomu ..... 230

*Przywoity model, lecz wciąż...* ..... 230

*Moment przełomowy* ..... 231

*Model pogłębiony* ..... 233

*Otrzeźwiająca decyzja* ..... 236

*Zaplata* ..... 237

Możliwości ..... 237

Koncentracja na podstawach ..... 237

Epilog — potok nowych spostrzeżeń ..... 238

### Rozdział 9. *Odkrywanie pojęć niejawnych* ..... 241

Wyciąganie pojęć ..... 242

*Nasłuchiwanie języka* ..... 242

*Analiza dziwnej implementacji* ..... 247

*Rozmyślanie nad sprzecznościami* ..... 252

*Czytanie książki* ..... 253

*Wielokrotne powtarzanie prób* ..... 255

W jaki sposób zamodelować mniej oczywiste pojęcia ..... 256

*Procesy jako obiekty dziedziny* ..... 259

*SPECYFIKACJA* ..... 261

*Zastosowanie SPECYFIKACJI w implementacji* ..... 264

### Rozdział 10. *Projekt elastyczny* ..... 279

*INTERFEJSY UJAWNIAJĄCE ZAMIAR* ..... 283

*FUNKCJE BEZ EFEKTÓW UBOCZNYCH* ..... 287

*ASERCJE* ..... 293

*Teraz widzimy lepiej* ..... 296

*ZARYSY KONCEPCYJNE* ..... 298

*Nieprzewidziana zmiana* ..... 301

*KLASY SAMODZIELNE* ..... 304

*ZAMKNIĘCIE OPERACJI* ..... 307

Projektowanie deklaratywne ..... 310

*Języki właściwe dziedzynie* ..... 311

Deklaratywny styl projektowania ..... 312

*Rozszerzenie SPECYFIKACJI w stylu deklaratywnym* ..... 313

*Subsumpcja* ..... 318

Kierunki ataku .....	321
<i>Definiowanie poddziedzin</i> .....	321
<i>W miarę możliwości polegaj na ustalonym formalizmie</i> .....	322
Rozdział 11. <i>Stosowanie wzorców analitycznych</i> .....	333
Rozdział 12. <i>Powiązanie wzorców projektowych z modelem</i> .....	349
STRATEGIA (zwana również POLITYKA) .....	351
KOMPOZYT .....	356
Dlaczego nie wzorec PYŁKU (FLYWEIGHT)? .....	361
Rozdział 13. <i>Refaktoryzacja ku głębszemu zrozumieniu</i> .....	363
Początek .....	364
Zespoły poszukiwawcze .....	364
Wczesniejsze odkrycia .....	365
Projekt dla programistów .....	366
Wycucie czasu .....	367
Kryzys jako źródło możliwości .....	368

## Część IV

Projekt strategiczny .....	369
----------------------------	-----

Rozdział 14. <i>Utrzymywanie integralności modelu</i> .....	373
KONTEKST ZWIĄZANY .....	377
<i>Rozpoznawanie odprysków pojęciowych</i>	
<i>w KONTEKŚCIE ZWIĄZANYM</i> .....	381
CIĄGŁA INTEGRACJA .....	383
MAPA KONTEKSTÓW .....	386
<i>Testowanie na granicach KONTEKSTU</i> .....	393
<i>Organizacja oraz dokumentacja MAP KONTEKSTÓW</i> .....	394
Relacje pomiędzy KONTEKSTAMI ZWIĄZANYMI .....	395
JĄDRO WSPÓLDZIELONE .....	396
ZESPOŁY PROGRAMISTYCZNE	
KLIENTA – DOSTAWCY .....	398
KONFORMISTA .....	403
WARSTWA ZAPOBIEGAJĄCA USZKODZENIU .....	406
<i>Projektowanie interfejsu WARSTWY ZAPOBIEGAJĄCEJ</i>	
USZKODZENIU .....	408
<i>Implementacja WARSTWY ZAPOBIEGAJĄCEJ</i>	
USZKODZENIU .....	408
<i>Opowieść ku przestrodze</i> .....	412

ODDZIELNE DROGI .....	414
USŁUGA OTWARTEGO GOSPODARZA .....	417
JĘZYK OPUBLIKOWANY .....	419
Unifikacja słonia .....	422
Wybór strategii kontekstu modelu .....	426
<i>Decyzja zespołowa lub wyższa</i> .....	426
<i>Stawianie siebie w kontekście</i> .....	427
<i>Przekształcanie granic</i> .....	427
<i>Akceptacja tego, czego nie możemy zmienić</i>	
— <i>wyznaczanie zewnętrznych systemów</i> .....	428
<i>Relacje z systemami zewnętrznymi</i> .....	429
<i>System w projektowaniu</i> .....	430
<i>Dostosowanie do specjalnych potrzeb przy użyciu</i>	
<i>odrębnych modeli</i> .....	431
<i>Wdrożenie</i> .....	432
<i>Kompromis</i> .....	433
<i>Kiedy projekt już trwa</i> .....	433
Transformacje .....	434
<i>Łączenie KONTEKSTÓW — ODDZIELNE DROGI →</i>	
<i>JĄDRO WSPÓLDZIELONE</i> .....	434
<i>Łączenie KONTEKSTÓW — JĄDRO</i>	
<i>WSPÓLDZIELONE → CIĄGŁA INTEGRACJA</i> .....	437
<i>Wygaszanie starego systemu</i> .....	438
<i>USŁUGA OTWARTEGO GOSPODARZA →</i>	
<i>JĘZYK OPUBLIKOWANY</i> .....	440
Rozdział 15. <i>Destylacja</i> .....	443
DZIEDZINA GŁÓWNA .....	446
<i>Wybór RDZENIA dziedziny</i> .....	449
<i>Kto wykonuje prace?</i> .....	449
Zwiększanie destylacji .....	451
PODDZIEDZINY OGÓLNE .....	452
<i>Ogólny nie oznacza możliwy do ponownego wykorzystania</i> .....	459
<i>Zarządzanie ryzykiem projektowym</i> .....	459
OPIS WIZJI DZIEDZINY .....	461
RDZEŃ WYRÓŻNIONY .....	464
<i>Dokument destylacji</i> .....	465
<i>RDZEŃ oznaczony</i> .....	466
<i>Dokument destylacji jako narzędzie procesowe</i> .....	467



SPÓJNE MECHANIZMY .....	469
OGÓLNE PODDZIEDZINY	
a SPÓJNE MECHANIZMY .....	471
<i>Kiedy MECHANIZM jest częścią</i>	
DZIEDZINY GŁÓWNEJ .....	472
Destylacja do stylu deklaratywnego .....	473
RDZEŃ ODDZIELONY .....	474
<i>Koszt utworzenia RDZENIA ODDZIELONEGO</i> .....	475
<i>Rozwijanie decyzji zespołowych</i> .....	476
RDZEŃ ABSTRAKCYJNY .....	481
Głęboka destylacja modelu .....	482
Wybór celów refaktoryzacji .....	483
 Rozdział 16. <i>Struktury dużej skali</i> .....	 485
PORZĄDEK EWOLUCYJNY .....	490
METAFORA SYSTEMU .....	493
<i>Dlaczego nie potrzebujemy metafory „naiwnej”</i> .....	495
WARSTWY ODPOWIEDZIALNOŚCI .....	496
<i>W jaki sposób struktura wpływa na bieżący projekt?</i> .....	502
<i>Wybór odpowiednich warstw</i> .....	505
POZIOM WIEDZY .....	511
SZKIELET KOMPONENTÓW DOŁĄCZANYCH .....	521
<i>Jak ograniczająca powinna być struktura?</i> .....	526
<i>Refaktoryzacja ku lepiej dopasowanej strukturze</i> .....	527
<i>Minimalizm</i> .....	528
<i>Komunikacja oraz samodyscyplina</i> .....	528
<i>Restrukturyzacja przyczynia się do projektu elastycznego</i> .....	529
<i>Destylacja zmniejsza obciążenie</i> .....	530
 Rozdział 17. <i>Łączenie strategii</i> .....	 531
<i>Łączenie struktur dużej skali</i>	
z KONTEKSTAMI ZWIĄZANYMI .....	531
<i>Łączenie struktur dużej skali oraz destylacji</i> .....	534
<i>Najpierw oszacowanie</i> .....	536
<i>Kto określa strategię?</i> .....	536
<i>Powstawanie struktury w trakcie tworzenia aplikacji</i> .....	537
<i>Zespół architektoniczny skoncentrowany na kliencie</i> .....	538
<i>Sześć podstawowych kryteriów dotyczących podejmowania</i>	
<i>strategicznych decyzji projektowych</i> .....	538
<i>To samo dotyczy szkieletów technicznych</i> .....	541
<i>Wystrzegaj się planu głównego</i> .....	542

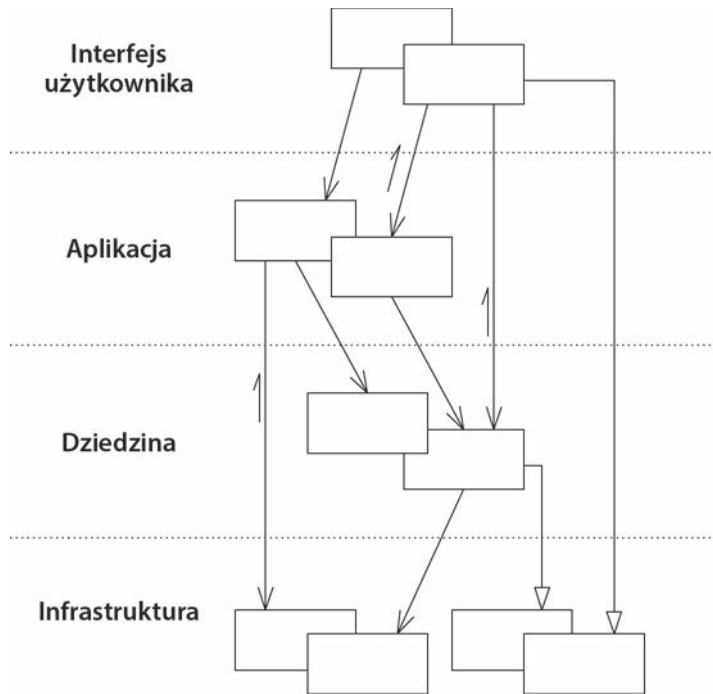
Zakończenie .....	545
Dodatek .....	553
Wykorzystanie szablonów z tej książki .....	553
Słownik .....	559
Bibliografia .....	565
Prawa do zdjęć .....	567
Skorowidz .....	569

# Wyizolowanie dziedziny

**F**ragment oprogramowania, który adresuje główne problemy dziedziny, stanowi najczęściej tylko małą część całego systemu informatycznego. Niemniej jednak jego znaczenie jest nieproporcjonalne w stosunku do rozmiaru. Aby móc zastosować nasze najlepsze pomysły, musimy być w stanie dostrzec w poszczególnych elementach modelu cały system. Nie możemy być zmuszeni do wybierania ich z większego zestawu obiektów, tak jak gwiazdozbiorów na nocnym niebie. Musimy oddzielić obiekty dziedziny od innych funkcji systemu, aby nie mieszać pojęć dziedzinowych z innymi, związanymi jedynie z technologią oprogramowania. W gąszczu całego systemu nie możemy też stracić z oczu samej dziedziny.

W tym celu powstały zaawansowane techniki służące do wyizolowania dziedziny. To dobrze znany obszar, lecz jest on tak istotny w prawidłowym zastosowaniu pryncypiów modelowania dziedzinowego, że musi zostać w tym miejscu krótko omówiony z punktu widzenia projektowania sterowanego dziedziną.

# Architektura warstwowa



W przypadku prostej aplikacji wspierającej użytkownika w procesie wyboru miejsca przeznaczenia dla ładunku z listy dostępnych miast musi istnieć kod programu, który (1) rysuje element interfejsu użytkownika na ekranie, (2) zadaje pytanie do bazy danych w celu uzyskania listy wszystkich możliwych miast, (3) odczytuje wybór użytkownika i go sprawdza, (4) przypisuje wybrane miasto do ładunku oraz (5) zapisuje wybór do bazy danych. Cały ten kod należy do jednego programu, lecz tylko jego mała część jest związana z dziedziną logistyki morskiej.

Programy wymagają od projektu oraz kodu, aby obsługiwały wiele różnych rodzajów zadań. Muszą przecież pobierać dane od użytkownika, wykonywać logikę biznesową, odwoływać się do bazy danych, komunikować przez sieci komputerowe, wyświetlać informacje dla użytkownika i tak dalej. Z tej przyczyny ilość kodu odpowiedzialnego za każdą funkcjonalność programu może być znaczna.

**W programowaniu obiektowym kod obsługujący interfejs użytkownika, bazę danych, a także wykonujący inne czynności pomocnicze jest bardzo często umieszczany bezpośrednio w obiektach biznesowych. Dodatkowa logika biznesowa jest również**

umieszczona w kodzie interfejsów użytkownika oraz skryptach bazodanowych. Dzieje się tak dlatego, iż w krótkiej perspektywie jest to najprostszy sposób na utworzenie działającego kodu.

W sytuacji, gdy kod związany z dziedziną jest rozproszony w tak dużej ilości innego kodu, niezmiernie trudno jest go dojrzeć i zrozumieć. Pobieżne zmiany do interfejsu użytkownika mogą w rzeczywistości zmienić logikę biznesową. A zmiana reguły biznesowej może wymagać skrupulatnego śledzenia kodu obsługującego interfejs użytkownika, bazę danych lub inne elementy programu. Implementacja spójnych obiektów sterowanych modelem staje się niepraktyczna, a testowanie automatyczne jest niewygodne. Z powodu wszystkich tych technologii i logiki zaangażowanych w każdą czynność program musi albo być bardzo prosty, albo też stanie się praktycznie niemożliwy do zrozumienia.

Tworzenie programów obsługujących bardzo złożone zadania wymaga rozdzielenia zagadnień, umożliwiając tym samym samodzielne i wyłączone skupienie się na różnych częściach projektu. W tym samym czasie złożone interakcje w systemie muszą być przeprowadzane z myślą o utrzymaniu tej rozłączności.

Istnieje wiele sposobów podziału systemu informatycznego, jednak doświadczenie przemysłowe pozwoliło utworzyć i zdefiniować *ARCHITEKTURY WARSTWOWE*, składające się w szczególności z kilku całkiem standardowych warstw. Metafora podziału systemu na warstwy jest tak powszechnie stosowana, że dla większości programistów wydaje się intuicyjna. W literaturze dostępnych jest wiele dobrych dyskusji na temat podziału systemów na warstwy. Niektóre z nich są nawet ujęte w postaci wzorców (jak w książce Buschmana<sup>1</sup> z roku 1996 — strony 31 – 51). Podstawowa zasada stanowi, że dowolny element w danej warstwie jest zależny jedynie od innych elementów w tej samej warstwie lub w warstwach niższych. Komunikacja w górę musi przechodzić przez mechanizm pośredni, który zostanie omówiony nieco później.

Podstawową wartością warstw jest fakt, iż każda z nich specjalizuje się w konkretnym aspekcie programu komputerowego. Ta specjalizacja umożliwia utworzenie bardziej spójnych projektów każdego z nich, a co za tym idzie, projekty te są prostsze w interpretacji. Oczywiście, niezmiernie istotne jest wybieranie warstw, które będą zawierać najbardziej spójne i ważne aspekty projektu. Z pomocą ponownie przychodzą doświadczenie

---

<sup>1</sup> Frank Buschman jest współautorem książki *Pattern-Oriented Software Architecture*, opisującej szczegółowo zastosowanie wzorców w projektowaniu architektury — *przyp. tłum.*

oraz konwencje przemysłowe. Chociaż istnieje wiele typów warstw, to jednak najbardziej skuteczne architektury używają pewnych odmian następujących czterech warstw pojęciowych:

<b>Warstwa interfejsu użytkownika (lub prezentacji)</b>	Odpowiedzialna za wyświetlanie informacji dla użytkownika oraz interpretację jego poleceń. Zamiast użytkownika zewnętrzny aktor może być niekiedy innym systemem komputerowym.
<b>Warstwa aplikacji</b>	Definiuje zamierzone zadania programu i steruje obiektami dziedziny w celu rozwiązania postawionych przed nimi zadań. Zadania, za które jest odpowiedzialna ta warstwa, są znaczące dla biznesu i wymagane w celu poprawnego współdziałania z warstwami aplikacji innych systemów. Warstwa jest utrzymywana w zwartej postaci. Nie zawiera reguł ani wiedzy biznesowej, a jedynie zarządza zadaniami i deleguje je do rozwiązania przez obiekty dziedziny w kolejnej warstwie niżej. Warstwa nie musi odzwierciedlać konkretnej sytuacji biznesowej, lecz może posiadać stan, ukazujący postęp zadania użytkownikowi lub programowi.
<b>Warstwa dziedziny (lub modelu)</b>	Warstwa odpowiedzialna za reprezentację zagadnień biznesowych, informacji o sytuacji biznesowej oraz reguł biznesowych. W tym miejscu jest przechowywany oraz używany stan odzwierciedlający sytuację biznesową, chociaż szczegóły techniczne związane z przechowywaniem tego stanu są oddelegowane do warstwy infrastruktury. <i>Ta warstwa stanowi istotę programu od strony biznesowej.</i>
<b>Warstwa infrastruktury</b>	Udostępnia podstawowe możliwości techniczne wspierające warstwy wyższe, tj. komunikaty wysyłane do aplikacji, zachowywanie dziedziny, rysowanie elementów interfejsu użytkownika itp. Warstwa infrastruktury może również poprzez szkielet architektury wspierać wzorzec interakcji pomiędzy wszystkimi czterema warstwami.

Niektóre projekty nie tworzą ostrego podziału pomiędzy warstwami interfejsu użytkownika oraz aplikacji. Inne mają wiele warstw infrastruktury. Jednak to właśnie wydzielenie *warstwy dziedziny* jest istotne w celu umożliwienia stosowania *PROJEKTU STEROWANEGO MODELEM*.

Dlatego:

**Podziel złożony program na warstwy. Utwórz projekt każdej z nich, czyniąc ją zarazem spójną, jak i zależną jedynie od warstw położonych niżej. Aby utrzymać luźne powiązanie z wyższymi warstwami, stosuj standardowe wzorce architektoniczne. Umieść kod związany z modelem dziedziny w pojedynczej warstwie i odizoluj go od kodu interfejsu użytkownika, aplikacji oraz infrastruktury. Dzięki takiemu podejściu obiekty dziedziny, pozbawione konieczności obsługi wyświetlania, przechowywania, zarządzania zadaniami aplikacji itd., mogą skoncentrować się jedynie na wyrażeniu modelu dziedziny. To umożliwi wzbogacenie i uproszczenie modelu na tyle, aby mógł on wychwycić wiedzę biznesową i wykorzystać ją w praktyce.**

Oddzielenie warstwy dziedziny od warstw infrastruktury oraz interfejsu użytkownika umożliwia utworzenie bardziej przejrzystego projektu każdej z nich. Utrzymanie wyizolowanych warstw jest zdecydowanie mniej kosztowne, ponieważ są one rozwijane we własnym tempie i odpowiadają na różne własne potrzeby. Podział na warstwy pomaga również we wdrożeniu systemu rozproszonego, pozwalając na umieszczenie poszczególnych warstw na różnych serwerach oraz klientach w celu zmniejszenia narzutu komunikacyjnego i zwiększenia wydajności (Fowler 1996).

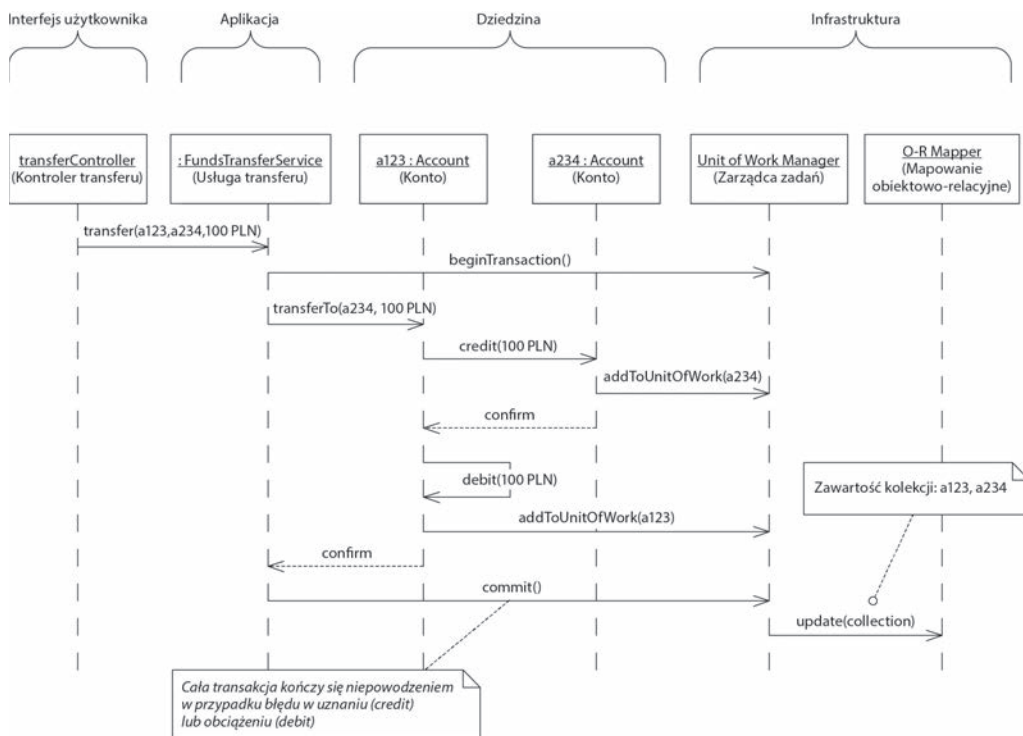
## Przykład

### Podział na warstwy w przypadku funkcjonalności internetowej systemu bankowego

Aplikacja udostępnia szereg funkcjonalności zarządzania kontami bankowymi. Jedną z nich jest przelew środków, gdzie użytkownik podaje numery dwóch rachunków oraz kwotę, a następnie zatwierdza przelew.

Aby ten przykład było łatwiej zrozumieć, pominę większość technicznych aspektów, a w szczególności zabezpieczenia. Projekt dziedziny zostanie również uproszczony (rzeczywista złożoność zwiększyłaby jedynie potrzebę posiadania architektury warstwowej). Co więcej, ta konkretna infrastruktura zaprezentowana w tym przykładzie miała w zamierzeniu być prosta oraz czytywista, co dotyczyć miało również samego przykładu — nie jest to żaden sugerowany projekt. Odpowiedzialności związane z pozostałymi funkcjonalnościami systemu będą podzielone na warstwy w sposób przedstawiony na rysunku 4.1.

Zauważ, że to warstwa dziedziny, a nie warstwa aplikacji jest odpowiedzialna za podstawowe reguły biznesowe — w tym przypadku reguła mówi: „Każde uznanie na rachunku musi mieć odpowiadające mu obciążenie”.



**Rysunek 4.1.**

Obiekty wypełniają odpowiedzialności zgodne z warstwą, do której przynależą, i są bardziej związane z innymi obiektami w tej samej warstwie

Aplikacja nie czyni żadnych założeń co do źródła pochodzenia żądania przelewu. Interfejs użytkownika zawierać będzie prawdopodobnie pola do wprowadzania numerów kont oraz wartości przelewu, a także przyciski odpowiedzialne za wydawanie poleceń. Mógłby on jednak — bez wpływu na warstwę aplikacji oraz jakkolwiek z niższych warstw — zostać zastąpiony żądaniem przelewu w postaci dokumentu XML. Takie rozdzielanie warstw występuje nie dlatego, że w projektach interfejsy użytkowników są często zastępowane dokumentami XML, lecz dlatego, że spójne rozdzielanie zadań czyni projekt każdej warstwy łatwym do zrozumienia i utrzymania.

W rzeczywistości sam rysunek 4.1 jedynie w sposób umiarkowany ilustruje problem braku izolacji dziedziny. Ponieważ musiał on obejmować wszystko, począwszy od żądania przelewu, do nadzorowania transakcji, warstwa dziedziny musi zostać uproszczona w taki sposób, aby cała interakcja z systemem była dość prosta do zrozumienia. Gdybyśmy skoncentrowali się na projekcie wyizolowanej warstwy dziedziny, zostawilibyśmy miejsce na stronie oraz w naszych głowach na model, który w lepszy sposób reprezentowałby reguły tej dziedziny. Być może dołączylibyśmy pełną rachunkowość, obiekty obciążeń i uznań rachunków albo też obiekty reprezentujące transakcje pieniężne.



## Powiązanie warstw

Jak dotąd nasza dyskusja skupiała się na rozdzieleniu warstw oraz na tym, w jaki sposób partycjonowanie ulepsza projekt każdego aspektu programu, w szczególności warstwy dziedziny. Jednak warstwy, oczywiście, muszą być ze sobą połączone. Wykonanie takiego połączenia bez utraty korzyści wynikających z ich rozdzielenia leży u podstaw wielu wzorców.

Warstwy mają być luźno związane, a zależności projektowe muszą być tylko jednokierunkowe. Warstwy wyższe mogą w prosty sposób używać elementów warstw niższych poprzez wykorzystanie ich interfejsów publicznych, przechowując odwołania do nich (choćby tymczasowo) — ogólnie rzecz biorąc, używając standardowych sposobów komunikacji. Jednakże w przypadku, gdy obiekt warstwy niższej potrzebuje skomunikować się w górę (nie mówimy o zwykłej odpowiedzi na zapytanie), musimy skorzystać z innego mechanizmu, używając w tym celu wzorca architektonicznego do łączenia warstw w rodzaju odwołań zwrotnych (*callbacks*) albo *OBSERWATORÓW* (patrz Gamma 1995 r.).

Dziadkiem wszystkich wzorców używanych do łączenia interfejsu użytkownika z warstwami aplikacji oraz dziedziny jest *MODEL-WIDOK-KONTROLER* (ang. *model-view-controler* — MVC). Wzorec ten został zapoczątkowany w okresie świetności Smalltalka w latach 70. ubiegłego wieku i zainspirował wiele późniejszych architektur interfejsu użytkownika. Wraz ze swoimi wieloma przydatnymi odmianami został on omówiony przez Fowlera (2003 r.). Również Larman (1998 r.) sprawdzał ten temat we *WZORCU ROZDZIELENIA MODELU-WIDOKU*, a opracowany przez niego *KOORDYNATOR APLIKACJI* jest jednym z podejść stosowanych do połączenia z warstwą aplikacji.

Oczywiście, istnieją inne style łączenia interfejsu użytkownika z aplikacją. Do naszych zastosowań wszystkie z nich będą poprawne, o ile będą zachowywać izolację warstwy dziedziny, pozwalając projektować obiekty dziedziny bez jednoczesnego przejmowania się interfejsem użytkownika, który mógłby ich później używać.

Warstwa infrastruktury nie inicjuje zazwyczaj żadnych akcji w warstwie dziedziny. Istniejąc „poniżej” warstwy dziedziny, nie powinna ona posiadać żadnej konkretnej wiedzy o dziedzinie, której służy. Rzeczywiście tego rodzaju techniczne możliwości są bardzo często udostępniane w charakterze *USŁUG*. Jeżeli na przykład aplikacja chce wysłać wiadomość e-mail, wtedy w warstwie infrastruktury może zostać zlokalizowany interfejs służący do wysyłania wiadomości, zaś elementy warstwy aplikacji mogłyby tylko zażądać jej wysłania. Tego rodzaju rozdzielenie daje dodatkową wielofunkcyjność. Interfejs do wysyłania wiadomości mógłby zostać

połączony z serwerem pocztowym, faksowym lub dowolnym innym aktualnie dostępnym. Jednak najważniejsza korzyść związana jest z uproszczeniem warstwy aplikacji, która jest wąsko skoncentrowana na swoim zadaniu i wie, *kiedy* ma wysłać wiadomość, lecz nie przejmuje się tym, *w jaki sposób* to wykonać.

Warstwy aplikacji oraz dziedziny polegają na *USŁUGACH* dostarczanych przez warstwę infrastruktury. Jeżeli tylko zakres *USŁUGI* został poprawnie wybrany, a jej interfejs prawidłowo zaprojektowany, wtedy obiekt wywołujący może pozostać luźno związany, bez uwzględniania skomplikowanej logiki kryjącej się pod interfejsem *USŁUGI*.

Jednak nie cała infrastruktura przykryta jest *USŁUGAMI* możliwymi do wywołania z warstw wyższych. Niektóre komponenty techniczne są zaprojektowane tak, aby w bezpośredni sposób wspierać podstawowe funkcje innych warstw (na przykład poprzez udostępnienie abstrakcyjnych klas bazowych dla obiektów dziedziny) oraz dostarczać mechanizm do ich powiązania (na przykład implementacje wzorca MVC i podobnych). Tego rodzaju „szkielet architektury” ma zdecydowanie większy wpływ na projekt innych części programu.

## Szkielety architektury

W sytuacji, gdy infrastruktura udostępniana jest w postaci *USŁUG* wywoływanych przez interfejsy, zrozumienie podziału na warstwy oraz sposobu ich utrzymania w sposób luźno związany ze sobą jest dość intuicyjne. Jednak niektóre problemy techniczne wymagają bardziej zachłannej formy infrastruktury. Struktury integrujące wiele potrzeb związanych z infrastrukturą wymagają często, aby inne warstwy były zaimplementowane w określony sposób, na przykład jako podklasa klasy szkieletowej lub z uporządkowanymi sygnaturami metod (wydawać by się mogło bardzo nieintuicyjne, aby podklasa była w warstwie wyższej niż klasa nadrzędna, ale miejmy na uwadze, która z klas zawiera więcej wiedzy o drugiej). Najlepszy szkielet architektury rozwiązuje złożone problemy techniczne, pozwalając programiście zajmującemu się dziedziną skoncentrować się na wyrażeniu modelu. Niemniej jednak szkielet ten może także stanąć na drodze programisty, tworząc na przykład zbyt wiele założeń ograniczających wybór projektu dziedziny lub też tak ciężką implementację, że będzie to opóźniać programowanie.

Pewnego rodzaju szkielet architektury jest zazwyczaj niezbędny (choć niekiedy zespoły wybierają taki, który niezbyt dobrze im służy). W trakcie stosowania szkieletu zespół musi skoncentrować się na swoim zadaniu, czyli stworzeniu implementacji wyrażającej model dziedziny, którego używa

do rozwiązania ważnych problemów. To właśnie zespół musi zaadaptować szkielet, nawet jeżeli będzie to oznaczać pominięcie pewnych jego funkcjonalności. Na przykład wczesne aplikacje J2EE bardzo często implementowały wszystkie obiekty dziedziny w postaci „ziaren encyjnych” (ang. *entity beans*). Takie podejście wpływało negatywnie zarówno na wydajność, jak i tempo programowania. Obecną praktyką jest raczej zastosowanie szkieletu J2EE do większych obiektów oraz implementacja większości logiki biznesowej przy użyciu prostych obiektów Javy. Wiele ograniczeń szkieletu architektury może być zminimalizowanych poprzez selektywne zastosowanie go do rozwiązania trudnych problemów, bez poszukiwania cudownego rozwiązania na wszystkie bolączki. Rozsądne stosowanie tylko wybranych, najbardziej wartościowych części rozwiązania zmniejsza związaną implementacją ze szkieletem architekturą, co daje większą elastyczność w późniejszych decyzjach projektowych. Biorąc pod uwagę, jak bardzo skomplikowanych jest wiele z obecnie dostępnych szkieletów aplikacji, podejście minimalistyczne w ich stosowaniu pomaga utrzymać obiekty biznesowe w przejrzystej i wymownej postaci.

Szkielety architektury oraz inne narzędzia wspomagające będą wciąż rozwijane. Nowsze z nich będą automatyzować i dostarczać wzorce dla coraz to większej liczby aspektów technicznych aplikacji. Jeżeli tylko zostaną one poprawnie zastosowane, programiści aplikacji będą mogli w coraz większym stopniu koncentrować się na modelowaniu podstawowych problemów biznesowych, co zwiększy produktywność zespołu oraz jakość pracy. Wybierając takie podejście, musimy wystrzegać się nadmiernego optymizmu w stosunku do gotowych rozwiązań technicznych — bardzo rozwlekłe szkielety mogą również ograniczać programistów.

## To w warstwie dziedziny żyje model

W większości dzisiejszych systemów stosowana jest *ARCHITEKTURA WARSTWOWA*, która używa wielu odmian podejścia do podziału na warstwy. Również wiele różnych stylów programowania może skorzystać z takiego podziału. Niemniej jednak projekt sterowany dziedziną wymaga istnienia tylko jednej konkretnej warstwy.

Model dziedziny jest zestawem pojęć. Wyrazem modelu oraz wszystkich związanych z nim bezpośrednio elementów dziedziny jest „warstwa dziedziny”. Składają się na nią dziedzina oraz implementacja logiki biznesowej. W *PROJEKCIE STEROWANYM MODELEM* struktury programu uzyskane w warstwie dziedziny odzwierciedlają pojęcia tego modelu.

Sytuacja, w której logika dziedziny jest wymieszana z innym kodem programu, nie jest zbyt praktyczna. Wyizolowanie dziedziny w procesie jej implementacji jest warunkiem wstępnym dla projektu sterowanego dziedziną.

## Antywzorzec inteligentnego interfejsu użytkownika

Tak właśnie wygląda powszechnie akceptowany wzorzec *ARCHITEKTURY WARSTWOWEJ* dla aplikacji obiektowych. Wprawdzie bardzo często próbuje się rozdzielić interfejsu użytkownika od aplikacji oraz dziedziny, jednak zbyt rzadko jest on w pełni osiągalny. Dlatego odstępstwa od tego wzorca wymagają odrębnej dyskusji.

Wiele projektów informatycznych podejmuje próbę stosowania znacznie mniej wyszukanego podejścia projektowego, nazywanego przeze mnie *INTELIGENTNYM INTERFEJSEM UŻYTKOWNIKA*. Jest to jednak tylko alternatywna, rozłączna odnoga w rozwoju, niezgodna z podejściem wymaganym przez projektowanie sterowane dziedziną. Jeżeli ktoś pójdzie tą drogą, większość treści tej książki nie będzie możliwa do zastosowania. Najczęściej interesują mnie sytuacje, w których wzorzec *INTELIGENTNEGO INTERFEJSU UŻYTKOWNIKA* nie powinien być stosowany, dlatego z przekąsem nazywam go „antywzorcem”. Omówienie go w tym miejscu będzie przydatnym orzeźwieniem i pomoże we wskazaniu przyczyn, które w dalszej części tej książki decydują o wyborze trudniejszej ścieżki projektowania.

\*\*\*

Projekt ma za zadanie dostarczyć prostą funkcjonalność, zdominowaną przez wprowadzanie oraz wyświetlanie danych, z jedynie kilkoma regułami biznesowymi. W skład zespołu projektowego nie wchodzi osoby z dużym doświadczeniem w modelowaniu obiektowym.

**W przypadku, gdy niedoświadczony zespół pracujący nad prostym projektem zdecyduje się wypróbować *PROJEKTOWANIE STEROWANE MODELEM z ARCHITEKTURĄ WARSTWOWĄ*, stanie przed koniecznością podjęcia szybkiej i wymagającej nauki. Członkowie zespołu będą zmuszeni opanować zaawansowane nowe technologie oraz dać sobie radę z nauką modelowania obiektowego (co jest wyzwaniem nawet z pomocą tej książki).**

Narzut spowodowany przez zarządzanie infrastrukturą oraz wszystkimi warstwami spowoduje wydłużenie prostych zadań. Proste projekty mają zazwyczaj krótkie terminy i średnio skomplikowane oczekiwania. Projekt zostanie z pewnością przerwany, na długo zanim zespół dokończy przypisane doń zadania, nie mając szans na zaprezentowanie wspaniałych możliwości zastosowanego podejścia.

Nawet jeżeli zespół będzie mieć więcej czasu, to bez pomocy eksperta członkowie zespołu prawdopodobnie nie dadzą rady opanować wymaganych technik. Jeżeli nawet na koniec pokonają te wszystkie przeciwności, to i tak w końcu utworzą prosty system, w którym nigdy nie będą wymagane bogate funkcjonalności.

Bardziej doświadczone zespoły nie będą musiały iść na takie kompromisy. Oczywiście, wieloletni programiści mogliby skrócić czas potrzebny na naukę i zarządzanie warstwami. Projektowanie sterowane dziedziną sprawdza się najlepiej w przypadku ambitnych projektów i będzie wymagać dużych umiejętności programistycznych. Nie wszystkie projekty są ambitne i nie wszystkie zespoły projektowe mogą opanować wymagane umiejętności.

Dlatego jeżeli tylko pozwalają na to okoliczności:

**Umieść całą logikę biznesową w interfejsie użytkownika. Podziel aplikację na małe funkcje i zaimplementuj je jako oddzielne moduły interfejsu, umieszczając w nich reguły biznesowe. Wykorzystaj bazę danych w charakterze współdzielonego repozytorium danych. Użyj najbardziej zautomatyzowanych narzędzi do tworzenia interfejsu użytkownika oraz programowania wizualnego, jakie tylko są dostępne na rynku.**

Herezja! Przecież dobra nowina mówi (i jest głoszona wszędzie, również w tej książce), że dziedzina powinna być oddzielona od interfejsu użytkownika. W rzeczywistości bez tego rozdzielenia trudno będzie zaimplementować jakąkolwiek metodę omawianą w tej książce, dlatego też wzorzec *INTELLIGENTNEGO INTERFEJSU UŻYTKOWNIKA* w kontekście projektowania sterowanego dziedziną może być traktowany jako „antywzorzec”. Jednak w niektórych sytuacjach skorzystanie z tego wzorca jest uprawnione. Mówiąc szczerze, ma on pewne zalety i w niektórych okolicznościach może sprawdzać się najlepiej — co częściowo tłumaczy, dlaczego jest tak popularny. Omówienie go w tym miejscu pomoże zrozumieć, dlaczego musimy oddzielać warstwę aplikacji od dziedziny i, co więcej, w jakich sytuacjach moglibyśmy nie chcieć tego robić.

### *Zalety:*

- Dostępna od razu duża wydajność dla prostych aplikacji.
- Mniej doświadczeni programiści mogą używać tego wzorca po pobieżnym przeszkoleniu.
- Nawet braki w wymaganiach mogą zostać rozwiązane po udostępnieniu prototypu użytkownikom, a następnie szybkim zaadaptowaniu go do ich potrzeb.
- Aplikacje są od siebie oddzielone, więc harmonogram dostarczania małych modułów może być opracowany z dość dużą dokładnością. Rozszerzenie systemu o dodatkowe proste funkcjonalności może być łatwe.
- Z tym wzorcem dobrze współpracują bazy danych, które umożliwiają integrację na poziomie danych.
- Dobra współpraca z narzędziami 4GL.
- Po udostępnieniu aplikacji osoby ją utrzymujące będą w stanie szybko zmienić jej części, nawet te, których nie będą w stanie zrozumieć. Ewentualne efekty zmian będą mieć tylko lokalny wpływ na konkretny fragment interfejsu użytkownika.

### *Wady:*

- Integracja aplikacji jest trudna, o ile nie jest zrobiona poprzez bazę danych.
- Nie występuje współdzielenie kodu oraz nie ma żadnego modelu abstrakcyjnego logiki biznesowej. Reguły biznesowe muszą być duplikowane w każdej operacji, której dotyczą.
- Szybkie prototypowanie oraz przeróbki kodu mają swoje naturalne ograniczenia, ponieważ brak abstrakcji dziediny ogranicza możliwości zmian.
- Złożoność szybko okaże się ograniczeniem nie do przecięcia. Dlatego naturalną ścieżką rozwoju będzie tworzenie kolejnych prostych aplikacji. Nie ma prostej recepty na wzbogacanie istniejącej logiki.

W przypadku świadomego zastosowania tego wzorca zespół może uniknąć dużego narzutu pracy związanej z innymi podejściami. Często pomyłką jest podejmowanie się korzystania ze złożonego podejścia do projektowania w sytuacji, gdy zespół nie jest gotów kontynuować go na całej ścieżce projektu. Kolejnym często popełnianym błędem jest tworzenie złożonej warstwy infrastrukturalnej i korzystanie z najlepszych narzędzi na rynku w projekcie, który właściwie tego nie potrzebuje.

Użycie większości uniwersalnych języków (takich jak Java) w przypadku tych aplikacji będzie stanowić dość dużą przesadę i będzie bardzo kosztowne. W takiej sytuacji skorzystanie z rozwiązań 4GL będzie najlepszym sposobem.

Należy jednak pamiętać, iż konsekwencją wyboru tego wzorca jest utrudniona migracja do innego podejścia projektowego — wybór ogranicza się właściwie do zastąpienia całych aplikacji nowymi. Samo użycie języków programowania ogólnego przeznaczenia, w rodzaju Javy, nie umożliwi w przyszłości łatwego porzucenia wzorca *INTELIGENTNEGO INTERFEJSU UŻYTKOWNIKA*. Jeżeli więc wybrałeś tę drogę, powinieneś również dostosować swoje narzędzia. Nie próbuj się asekurować. Z samego zastosowania uniwersalnego języka nie będzie wynikać elastyczność systemu. Może się to jednak skończyć zwiększeniem kosztów.

Analogicznie zespół, który wybrał drogę *PROJEKTOWANIA STEROWANEGO MODELEM*, powinien dostosować swój projekt od samego początku. Oczywiście, nawet bardzo doświadczone, ambitne zespoły projektowe muszą rozpocząć od prostej funkcjonalności i rozwijać swą pracę w kolejnych etapach. Jednakże te pierwsze czynności muszą być *STEROWANE MODELEM* z wydzieloną warstwą dziedziny, gdyż w przeciwnym razie projekt prawdopodobnie skończy na podejściu wykorzystującym *INTELIGENTNY INTERFEJS UŻYTKOWNIKA*.

\*\*\*

Wzorzec *INTELIGENTNEGO INTERFEJSU UŻYTKOWNIKA* został tutaj omówiony jedynie po to, aby wyjaśnić przyczyny sytuacji, w których wzorzec *ARCHITEKTURY WARSTWOWEJ* jest niezbędny do wyizolowania warstwy dziedziny.

Oczywiście, istnieją inne rozwiązania, leżące pomiędzy omówionymi w tym rozdziale wzorcami. Na przykład Fowler w swojej książce opisuje *SKRYPT TRANSAKCYJNY*, który oddziela interfejs użytkownika od aplikacji, jednak nie udostępnia modelu obiektowego. Płyne z tego następujący wniosek: *Jeżeli aplikacja wydziela kod związany z dziedziną w postaci spójnego projektu dziedziny luźno związanego z pozostałą częścią systemu, wtedy taka architektura prawdopodobnie mogłaby zostać zmodyfikowana do postaci projektu sterowanego dziedziną.*

Każdy z innych stylów programowania ma swoje zastosowanie, a Ty, drogi Czytelniku, musisz nauczyć się akceptować kompromis pomiędzy złożonością a elastycznością. W niektórych sytuacjach brak wydzielenia projektu dziedziny może być naprawdę tragiczny. Jeżeli masz złożoną aplikację i zdecydowałeś się na *PROJEKT STEROWANY MODELEM*, wytrwaj w tym do końca, zatrudnij niezbędnych ekspertów i unikaj *INTELIGENTNEGO INTERFEJSU UŻYTKOWNIKA*.

## Inne rodzaje izolacji

Niestety, poza użytkownikiem oraz infrastrukturą istnieje wiele innych czynników, które mogą zepsuć delikatny model dziedziny. Będziesz musiał radzić sobie z innymi elementami dziedziny, które nie będą w pełni zintegrowane z Twoim modelem. Będziesz musiał także współpracować z innymi zespołami programistów, które będą wykorzystywać inne modele tej samej dziedziny. To wszystko będą czynniki wpływające na zaciemnienie modelu oraz zmniejszenie jego przydatności. Tymi zagadnieniami zajmiemy się w rozdziale 14., „Utrzymywanie integralności modelu”, w którym wprowadzimy takie wzorce jak *KONTEKST ZWIĄZANY* oraz *WARSTWA OCHRONY PRZED USZKODZENIEM* (ang. *anticorruption layer*). Bardzo złożony model dziedziny może sam stać się bezużyteczny. W rozdziale 15., „Destylacja”, zajmiemy się omówieniem metod umożliwiających warstwie dziedziny oddzielenie pojęć zasadniczych od tych wspierających.

Tym wszystkim zajmiemy się jednak później. W kolejnym rozdziale natomiast przyjrzymy się podstawom umożliwiającym wspólny rozwój efektywnego modelu dziedziny wraz z jego funkcjonalną implementacją. Ostatecznie największym zyskiem z jej wydzielenia jest usunięcie wszystkich pozostałych rzeczy z drogi, co ułatwia rzeczywiste skoncentrowanie się na projektowaniu dziedziny.



---

# SKOROWIDZ

---

## A

abstract factory, 168  
ADAPTER, 411  
agile, 51  
AGREGATY, 153, 155–165, 203  
akceptacja, 429  
aktywa, 254  
analiza  
    dziwnej implementacji, 247  
    zysków, 402  
antywzorzec inteligentnego  
    interfejsu użytkownika, 102  
ARCHITEKTURA WARSTWOWA, 47,  
    95, 101, 142, 443  
ASERCJE, 286, 293–297  
asocjacje, 109, 111, 201, 258  
asocjacje pomiędzy klasami, 113  
atrybuty, 117, 201

## B

baza danych, 130  
BUDOWNICZY, 169  
builder, 169

## C

cechy USŁUGI, 133  
cel operacji logistycznej, 197  
chemiczny JĘZYK OPUBLIKOWANY, 422  
CIĄGŁA INTEGRACJA, 376, 383, 429, 438  
ciągła nauka, 44  
cykl życia obiektu dziedziny, 153, 154  
cykliczna referencja, 202  
czyszczenie pamięci, 129, 153

## D

decyzje  
    proaktywne, 375  
    projektowe, 538  
    zespołowe, 427

definiowanie

    obiektów, 107  
    poddziedzin, 321  
    tożsamości, 121  
deklaratywny styl projektowania, 312  
destylacja, 443, 451, 473, 530, 534  
destylacja  
    modelu, 41  
    strategiczna, 445  
diagram UML, 36, 63, 65  
    interakcji, 64  
    klas, 39, 46, 70  
    sekwencji, 64, 346  
dialekt, 53  
dodanie  
    obiektu, 210  
    operacji obsługi, 208  
dokument, 63, 66, 67  
    destylacji, 465, 467  
    XML, 98  
dokumentacja MAP KONTEKSTÓW, 394  
dokumenty projektowe, 65  
dostęp do  
    bazy danych, 143, 144  
    USŁUG, 137  
    WARTOŚCI, 181  
dostosowanie, 432  
duża spójność, high-cohesion, 108  
dystrybucja spłaty kapitału, 233  
dziedzina, 30, 93  
DZIEDZINY GŁÓWNE, 444, 446, 458, 472

## E

efekty uboczne metody, 289  
eksperti dziedzinowi, 61  
elastyczność projektu, 280, 328  
elementy  
    składowe projektu, 37, 38, 89  
    wydajnego modelowania, 40  
ENCJE, 45, 115–122, 125–130, 158, 199

## F

FABRYKA ABSTRAKCYJNA, 168  
FABRYKI, 154, 166–177  
    ENCJI, 174  
    WARTOŚCI, 174  
factory method, 168  
faktoryzacja OGÓLNYCH  
    PODDZIEDZIN, 473  
fałszywe pokrewieństwa, 381  
FASADA, 410  
funkcja, 287  
    calculateAccrualsThroughDate(), 251  
FUNKCJE BEZ EFEKTÓW  
    UBOCZNYCH, 286–288, 297  
funkcjonalność  
    powtórzenie operacji, 206  
    sprawdzanie przydziału, 215  
    zmiana miejsca przeznaczenia  
        ładunku, 206

## G

garbage collection, 153  
garbage collector, 129  
generowanie, 270  
GENERYCZNE PODDOMENY, 286  
głęboka destylacja modelu, 482  
granice AGREGATU, 203

## I

idea JĘZYKA WSZECHOBECNEGO, 55  
identyfikator, 207  
identyfikator obiektu, 119  
implementacja, 20  
    asocjacji, 109  
    ENCJI, 117  
    kolekcji, 212  
    REPOZYTORIÓW, 188  
    REPOZYTORIUM, 185, 186  
    WARSTWY ZAPOBIEGAJĄCEJ  
        USZKODZENIU, 409  
informacje o projekcie, 65  
inicjalizacja Itinerary, 58  
inicjalizator, 288  
integracja wzorców, 322  
integralność  
    modelu, 373  
    procesu zakupowego, 160  
INTELIGENTNY INTERFEJS  
    UŻYTKOWNIKA, 102, 105

interakcje z FABRYKĄ, 168  
INTERFEJS, 173, 283  
    USŁUGI, 100  
        użytkownika, 96, 102  
INTERFEJSY UJAWNIAJĄCE ZAMIAR,  
    286, 297  
interpretacja asocjacji, 109  
inwestycja pożyczkowa, 230  
istota programu, 32  
izolacja, 106  
izolacja dziedziny, 98, 198

## J

JĄDRO WSPÓLDZIELONE, 376,  
    395–398, 406, 435, 438  
JĘZYK, 53, 61  
    definiowany modelem, 41  
    Java, 113  
    mówiony, 58  
    naturalny, 70  
    oparty na modelu, 61  
    OPUBLIKOWANY, 420–423, 441  
    pidżynowy, 59  
    UML, 63  
    WSZECHOBECNY, 53, 60, 67, 146, 192  
języki właściwe dziedzinie, 311

## K

kalkulator, 254  
kaskada spostrzeżeń, 326  
kierunek asocjacji, 110  
klasa Enterprise, 370  
KLASY SAMODZIELNE, 304–306  
kodowanie pakietów, 141  
KOMPOZYT, 315, 356–361  
kompromis, 434  
komunikacja, 51, 528  
koncepcja trasy, 358  
KONFORMISTA, 404, 405  
konstruktor, 207  
konstruktory publiczne, 172  
KONTEKST, 425, 430  
    rezerwacji, 379  
    ZWIĄZANY, 55, 106, 372, 376–382,  
        390, 394, 428, 488, 521, 527, 532  
KOORDYNATOR APLIKACJI, 99  
korygowanie pożyczki, 235  
korzeń AGREGATU, 188, 206  
koszt utworzenia RDZENIA  
    ODDZIELONEGO, 475

kryzys, 368  
kwalifikacja asocjacji, 113  
kwalifikator, 109

## L

lista płac, 519  
logika  
    biznesowa, 94  
    niezmienników, 174  
logistyka morska, 213

## Ł

łączenie  
    dwóch systemów, 216  
    FABRYKI z REPOZYTORIUM, 190  
    KONTEKSTÓW, 435, 438  
    paradygmatów, 150  
    SPECYFIKACJI, 313  
    strategii, 531  
    struktur dużej skali, 531, 534  
    warstw, 99

## M

magazyn obiektów, 191  
MAPA KONTEKSTÓW, 376, 386–395,  
    427, 534, 549  
mapa nawigacyjna, 91, 445  
mapowanie  
    metadanych, 182  
    obiektowo-relacyjne, 267  
MECHANIZM, 472  
    SPÓJNOŚCI, 286  
    w schemacie organizacyjnym, 470  
metafora naiwna, 495  
METAFORY SYSTEMU, 493, 495  
metoda  
    anInvoice.isOverdue(), 261  
    asSql(), 269  
    dostępu, accessor method, 109  
    isOverdue(), 261  
    isSafelyPacked(), 274  
    mixIn(), 289  
METODY  
    FABRYCZNE, 209  
    refaktoringu, 42  
    testujące, 297  
    WYTWÓRCZE, 168, 172, 209  
metodyka, 65  
metodyka zwinna, agile, 51

mieszanie farb, 284, 295  
mikrorefaktoryzacja, 225  
minimalizm, 528  
minimalna abstrakcja dziedziny, 56  
model, 31, 101  
    dziedziny, 31, 51, 101  
    firmy spedycyjnej, 243  
    GŁÓWNY, 549  
    konta inwestycyjnego, 111  
    księgowy, 336  
    logistyki, 213  
    pogłębiony, 233  
    pożyczki, 234  
    transportu morskiego, 476  
    UML, 63, 68  
    wzbogacony wiedzą, 41, 57  
    zrefaktoryzowany, 503  
    zrestrukturyzowany, 503

## modele

    dogłębne, 48, 226, 227  
    dziedzinowe, 351  
    księgowe, 336  
    obiektove, 149  
    objaśniające, 68

## modelowanie

    AGREGATÓW, 154, 212  
    dogłębne, 241  
    ENCJI, 120, 212  
    na głos, 58  
    obiektove, 117  
    pojęć, 256, 283  
    USŁUGI, 136  
    WARTOŚCI, 212  
    wydajne, 40  
MODEL-WIDOK-KONTROLER, 99  
modularność, 145  
MODUŁ  
MODUŁ RDZENIA, 481  
MODUŁY, 108, 138–141, 213, 487  
    DZIEDZINY GŁÓWNEJ, 535  
    zwinne, 140  
moment przełomowy, 229, 231  
MVC, model-view-controller, 99

## N

nadkomplet, 258  
naliczanie odsetek, 335  
naruszony niezmiennik, 162  
narzędzie QueryService, 112  
narzut, 375

- nasłuchiwanie
    - brakującego pojęcia, 243
    - języka, 242
  - nazwa
    - klasy, 284
    - WZORCA, 557
    - MODUŁU, 213
  - nicobiektowy, 149
  - niezmiennik, 128, 164, 174, 256
  - niezmienniki AGREGATU, 159
  - normalizacja, 192
  - notacja UML, 51
  - numer
    - PESEL, 123
    - Social Security, 123
- O**
- obiekt, 108
    - Brokerage Account, 111, 170
    - Cargo, 197, 199
    - Catalog Part, 173
    - Charge, 373
    - Container, 274
    - Customer, 199
    - Delivery History, 198
    - Delivery Specification, 197, 198
    - Facility, 231, 238
    - Invoice, 263
    - Itinerary, 58
    - Loan, 238
    - Location, 200
    - Packer, 275
    - Paint, 291
    - pojęciowy, 144
    - REPOSITORY, 268
    - Route Specification, 58
    - Routing Service, 58
    - Share Pic, 238, 325
    - SharePie, 326, 329
    - SPECIFICATION, 268
    - Strategia, 221
    - Trade Order, 170
    - typu Cargo, 46
    - typu Voyage, 46
  - obiekty
    - dziedziny, 259
    - niezmiennicze, 128
    - referencyjne, 115
    - WARTOŚCI, 128
    - złożone, 168
  - OBSERWATOR, 99
  - obsługa XML, 423
  - oczekiwania komponentu podrzędnego, 402
  - oddelegowanie implementacji, 455
  - oddzielanie
    - poleczeń, 324
    - RDZENIA, 476
    - warstwy aplikacji od dziedziny, 103
  - ODDZIELNE DROGI, 376, 415–417, 435
  - odkrywanie pojęć niejawnych, 241
  - odnajdowanie tras, 352
  - odpowiedzialności
    - dziedziny, 219
    - operacyjne, 498
    - potencjału, 499
    - wsparcia decyzji, 500
  - odpytywanie REPOZYTORIUM, 184
  - odsetki, 247, 335
  - odszukiwanie, 266
  - odtworzenie obiektu, 175–177
  - odwołania zwrotne, 99
  - OGÓLNE PODDZIEDZINY, 454, 459, 471
  - ograniczenia, 257, 259
    - asocjacji, 110, 114
    - kierunku interpretacji, 111
  - określanie strategii, 536
  - operacje, 507
    - obsługi, 208
    - spedycyjne, 200
    - na tożsamości, 121
  - operator
    - AND, 316, 318
    - NOT, 318
  - OPIS WIZJI, 468
  - OPIS WIZJI DZIEDZINY, 461, 462, 463
  - opowiadanie historii, 506
  - optymalizacja bazy danych, 130
  - opublikowany projekt, 454
  - organizacja, 394
  - osadzanie reguły, 268
  - oszacowanie, 536
  - OTWARTE USŁUGI GOSPODARZA, 395
- P**
- PAKIETY, 138
  - pakowacz, 272, 275
  - paradygmat
    - obiektowy, 146
    - relacyjny, 152
  - paradygmaty modelowania, 146
  - partycjonowanie, 144

PCB, printed-circuit board, 35  
plan główny, 542  
początkowy stan zamówienia, 161  
PODDZIEDZINY, 322, 535  
PODDZIEDZINY OGÓLNE, 444, 452  
podejmowanie decyzji, 539  
podział  
na pakiety, 144  
na partycje, 144  
na warstwy, 97, 498  
systemu informatycznego, 95  
podział usług, 136  
pojęcia  
ukryte, 325  
wysokopoziomowe, 283  
polimorfizm, 171  
polityka, policy, 47, 351, 508  
nadkompletu, 48, 258  
trasowania, 352, 503  
połączenie modelu z implementacją, 107  
poprawa wydajności, 219  
PORZĄDEK EWOLUCYJNY, 490, 505  
poszukiwanie pojęć, 253  
potencjał, 507  
powiązanie  
warstw, 99  
wzorców projektowych, 349  
powtarzanie prób, 255  
POZIOM WIEDZY, 511–520  
poziomy refraktoryzacji, 224  
pozycja, 238  
pożyczka, 234  
predykat, 261  
proces, 259  
odkrywania, 228  
projektowania, 20  
XP, 22  
programowanie ekstremalne, 21, 66  
programowanie  
ekstremalne, XP, 21  
obiektywne, 94  
wizualne, 103  
projekt  
AGREGATU, 209  
dla programistów, 366  
dystrybucji płatności, 323  
elastyczny, 279  
GŁÓWNEJ DZIEDZINY, 371  
STEROWANY MODELEM, 89, 101,  
107, 146, 151, 264  
strategiczny, 369, 540  
ubezpieczeń, 416  
projektowanie  
asocjacji, 131, 201  
deklaratywne, 310, 311  
interfejsu, 173, 409  
kontraktowe, 90  
modeli, 224  
obiektów, 190  
OBIEKTÓW WARTOŚCI, 128  
obwodów drukowanych, 35  
operacji, 121  
sterowane dziedziną, 17, 31, 93, 283  
STEROWANE MODELEM, 68, 91, 105  
STEROWANE MODELEM  
z ARCHITEKTURĄ  
WARSTWOWĄ, 102  
sterowane odpowiedzialnością, 90  
projekty elastyczne, 227, 529  
prototyp, 277  
prototyp pakowacza, 275  
przechowywanie danych, 150  
przeglądanie scenariuszy, 206  
przekształcanie granic, 428  
przełom, 229–231  
przepływ  
informacji, 216  
zadań, 150  
przetwarzanie  
wiedzy, 35, 41  
wsadowe, 341

## R

RDZEŃ, 458  
ABSTRAKCYJNY, 481, 521, 548  
DZIEDZINY, 449  
ODDZIELONY, 474–479  
oznaczony, 466  
WYRÓŻNIONY, 464–468  
realizacja produkcji, 523  
refaktoring, 167, 209  
implementacji, 45  
modelu obiektowego, 193  
refaktoryzacja, 223, 236, 284, 363–368,  
483, 527  
aplikacji, 289  
kodu, 261  
obiektu, 290  
ograniczenia, 257  
referencja, 178, 202  
referencje do obiektów, 158  
REFLEKSJA, 516  
reguła wymagalności, 264

- reguły
    - biznesowe, 262
    - księgowania, 341, 346
    - łączenia nieobiektywnych elementów, 151
    - organizacji, 515
  - relacje
    - kwalifikowane, 110
    - między elementami
      - AGREGATU, 158
    - między KONTEKSTAMI
      - ZWIĄZANYMI, 395
    - z FABRYKAMI, 189
  - relacyjne bazy danych, 112, 190, 267
  - REPOZYTORIA, 154, 178–190, 202, 269
  - repozytorium faktur, 266
  - restrukturyzacja, 529
  - rezerwacje, 402, 413
  - RGB, 290
  - rodzaj wiedzy, 45
  - rodzaje
    - działalności, 508
    - izolacji, 106
  - rola, 197, 201
  - Routing Service, 56
  - rozdzielanie zagadnień, 95
  - rozdzielenie warstw, 98
  - rozpoznawanie odprysków pojęciowych, 381
  - rozszerzanie ENCJI oraz WARTOŚCI, 199
  - rozszerzalny język znaczników, 421
  - rozszerzenie
    - AGREGATU, 170
    - języka, 62
    - SPECYFIKACJI, 313
  - rozwijanie decyzji zespołowych, 476
  - ryzyko projektowe, 459
- S**
- samodyscyplina, 528
  - samodzielna implementacja, 455
  - scenariusz refaktoryzacji, 363
  - scenariusze, 206
  - schemat organizacyjny, 470, 472
  - SEGMENT PRZEDSIĘBIORSTWA, 219
  - segmentacja biznesu, 217
  - serializacja, 122
  - sieć ścieżek, 37
  - silniki reguł biznesowych, 145, 150
  - SINGLETON, 137
  - SKRYPT TRANSAKCYJNY, 105
  - słabe związanie, loose-coupling, 108
  - SPECYFIKACJA, 185, 197, 260–274, 315, 317, 319
    - Arystotelesa, 321
    - KOMPOZYTU, 317
  - SPÓJNE MECHANIZMY, 469, 471, 473
  - spójność zmian, 156
  - SPÓJNY MECHANIZM, 469–471
  - sprzeczności, 252
  - SQL, 113, 268
  - standaryzowany produkt, 453
  - stosowanie wzorców analitycznych, 333
  - STRATEGIA, 47, 351–355, 536
  - strefy czasowe, 458
  - struktura, 502, 504, 526, 537
  - struktury dużej skali, 485, 524
  - styl deklaracyjny, 473
  - subsumpcja, 318, 320
  - system
    - automatyzacji fabryki, 509
    - bankowości inwestycyjnej, 510
    - do wypłat pracowniczych, 518
    - do wypłat pracowniczych, 512
    - logistyczny, 195
    - logistyki morskiej, 498
    - obsługi zamówienia, 160
    - w projektowaniu, 431
    - zewnętrzny, 430
  - systemy
    - klasy Enterprise, 370
    - wsparcia decyzji, 508
  - szablony, 553
  - szkielet
    - architektury, 100, 188
    - CIM, 524
    - języka, 54
    - KOMPONENTÓW
      - DOŁĄCZANYCH, 521, 524
      - SEMATECH CIM, 523
      - techniczny, 541
- Ś**
- śledzenie
    - płatności, 116
    - tożsamości ENCJI, 126
- T**
- testowanie, 297, 394
    - języka, 54
    - trasy, 391
  - topologia, 37, 39

- tożsamość, 121  
 ENCJI, 117, 126  
 globalna, 158  
 lokalna, 158  
 transakcje, 238  
 transformacje, 435  
 trasowanie, 353, 503  
 tworzenie  
 AGREGATU, 171  
 elementu narzuty, 525  
 harmonogramów, 456  
 interfejsu użytkownika, 103  
 modelu, 36  
 modelu dziedziny, 91  
 na zamówienie, 270  
 obiektów, 167, 207  
 obiektów złożonych, 168  
 pakietów, 142  
 RDZENIA ODDZIELONEGO, 475  
 REPOZYTORIUM, 211
- ## U
- ujawnianie ukrytych pojęć, 325  
 ukryte pojęcie, 45  
 UML, Unified Modeling Language, 51, 63  
 unifikacja, 375  
 UPORZĄDKOWANIE EWOLUCYJNE, 510  
 USŁUGA, service, 99, 107, 132–135, 217  
 aplikacyjna, 136  
 dziedzina, 136  
 infrastrukturalna, 136  
 OTWARTEGO GOSPODARZA, 418, 420, 441  
 trasowania, 56, 353  
 usuwanie  
 asocjacji, 109  
 zatorów projektowych, 277  
 utrzymywanie integralności, 373  
 używanie  
 języka, 51, 195  
 JĘZYKA WSZECHOBECNEGO, 54, 62
- ## W
- walidacja, 265  
 warstwa, 95, 97  
 aplikacji, 96, 97  
 dziedziny, 96, 97, 101, 134  
 infrastruktury, 96, 99  
 interfejsu użytkownika, 96, 97  
 OCHRONY PRZED  
 USZKODZENIEM, 106  
 potencjału, 509  
 prezentacji, 96  
 ZAPOBIEGAJĄCA  
 USZKODZENIU, 218, 407–414, 430  
 zobowiązania, 509  
 warstwy  
 powiązania, 99  
 DZIEDZINY, 267  
 MAPOWANIA METADANYCH, 180  
 ODPOWIEDZIALNOŚCI, 496, 505, 526, 533  
 WARTOŚCI, 125–131, 181  
 WARTOŚĆ, value-object, 107  
 warunki brzegowe, 59  
 wdrożenie, 433  
 wiedza biznesowa, 164  
 wielorakość relacji, 109  
 WIZJA DZIEDZINY, 461  
 wprowadzenie aplikacji, 198  
 wsparcie decyzji, 508  
 współdzielenie  
 bazy danych, 161  
 obiektów, 128  
 wybór, 266  
 celów refaktoryzacji, 483  
 FABRYK, 169  
 RDZENIA dziedziny, 449  
 REPOZYTORIÓW, 204  
 strategii kontekstu modelu, 427  
 warstw, 505  
 z kolekcji, 308  
 wyciąganie pojęć, 242  
 wyczucie czasu, 367  
 wydajność, 219  
 wydobywanie ukrytego pojęcia, 45  
 wygaszanie systemu, 439  
 wyizolowanie dziedziny, 93  
 wykonywanie reguł księgowania, 342  
 wymagania, 434  
 wymagania specjalizowane, 375  
 wymogi infrastruktury, 142  
 wymuszanie  
 niezmienników, 161  
 niezmiennika AGREGATU, 165  
 wyświetlanie informacji, 96  
 wyznaczanie trasy ładunku, 499  
 wzbogacony model dziedziny, 57

- wzorce
    - analityczne, 217, 333, 346
    - destylacyjne, 460
    - elementów modelu, 107
    - projektowe, 349
  - wzorzec
    - ARCHITEKTURY WARSTWOWEJ, 102
    - INTELIGENTNEGO INTERFEJSU UŻYTKOWNIKA, 103, 105
    - PYŁKU, 361
    - REPOZYTORIUM, 182
    - ROZDZIELENIA MODELU-WIDOKU, 99
    - SEGMENT PRZEDSIĘBIORSTWA, 219
    - SINGLETON, 137
    - PECYFIKACJI, 197
    - STRATEGIA, 475
- X**
- XP, extreme programming, 21
- Z**
- zablokowanie zamówienia, 163
  - zachowanie obiektu, 64
  - zakleszczenie, 164
  - zakres USŁUGI, 100
  - zalety REPOZYTORIÓW, 183
  - zależności
    - pojęciowe, 506, 509, 510
    - projektowe, 99
  - ZAMKNIĘCIE OPERACJI, 307–310, 327
  - zapłata, 237
  - zapytanie, 179, 182, 184
    - oparte na SPECYFIKACJI, 185
    - SQL, 268
  - ZARYSY
    - KONCEPCYJNE, 298–303, 507, 529
    - sum przyrostowych, 300
  - zarządzanie
    - kontami bankowymi, 97
    - obiektami, 153
    - ryzykiem projektowym, 459
    - zmianą, 161
  - zasada podwójnego księgowania, 337
  - zastosowanie
    - modelu dziedziny, 29
    - SPECYFIKACJI, 264
  - zduplikowane pojęcia, 381
  - ZESPOŁY PROGRAMISTYCZNE KLIENTA – DOSTAWCY, 399
  - zespół, 60
  - zespół architektoniczny, 538
  - ziarna encyjne, 101
  - ziarnistość, 136
  - złożone funkcjonalności, 19
  - zmiana
    - JĘZYKA WSZECHOBECNEGO, 55
    - refaktoryzacyjna, 237
  - zmienna logiczna, 56
  - zwiększanie destylacji, 451
  - zwrot z refaktoryzacji, 229



# PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW  
w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

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

GRUPA WYDAWNICZA

 **Helion SA**

**Tworzenie skomplikowanych systemów informatycznych** wymaga nowego podejścia. Dotychczas stosowane metody przestają się sprawdzać i generują mnóstwo problemów. Odpowiedzią na nie jest Domain-Driven Design, w skrócie DDD. W tym podejściu szczególnie nacisk kładzie się na tworzenie obiektów dokładnie odzwierciedlających zachowanie ich odpowiedników istniejących w rzeczywistości. Dzięki temu projektowanie systemu można powierzyć ekspertom z danej branży, którzy niekoniecznie muszą być specjalistami w dziedzinie projektowania architektury systemów informatycznych.

**Ta książka jest niezwykłym przewodnikiem**, który wprowadzi Cię w świat DDD. Sięgnij po nią i poznaj elementy składowe projektu sterowanego modelem oraz cykl życia obiektu dziedziny. W trakcie lektury kolejnych rozdziałów dowiesz się, jak odkrywać pojęcia niejawne, stosować wzorce analityczne oraz wiązać projekty z modelem. Ponadto zobaczysz, w jaki sposób utrzymywać integralność modelu, a na sam koniec zaznajomisz się ze strukturami dużej skali oraz złączeniem strategii. Ta książka jest doskonałą lekturą dla wszystkich osób chcących zrozumieć Domain-Driven Design oraz zastosować to podejście w praktyce!

### Dzięki tej książce:

- zrozumiesz ideę Domain-Driven Design
- nauczysz się tworzyć modele
- zadbasz o integralność stworzonego modelu
- uporządkujesz system za pomocą struktur dużej skali
- rozpoznasz momenty przełomowe w trakcie modelowania oraz na nie zareagujesz
- wykorzystasz DDD w Twoim projekcie

## Sprawdź, jak projektować skomplikowane systemy informatyczne!

**ERIC EVANS** — uznawany za czołowego projektanta systemów informatycznych, założyciel Domain Language. Pracował przy projektach tworzonych w językach Java oraz Smalltalk dla branży finansowej, ubezpieczeniowej oraz transportowej.

**Helion**

33517 numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:  
• <http://helion.pl/promocje>  
Książki najchętniej czytane:  
• <http://helion.pl/bestsellery>  
Zamów informacje o nowościach:  
• <http://helion.pl/nowosci>

Helion SA  
ul. Kościuszki 1c, 44-100 Gliwice  
tel.: 32 230 98 63  
e-mail: [helion@helion.pl](mailto:helion@helion.pl)  
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-0525-0



9 788328 305250

Informatyka w najlepszym wydaniu

cena: 99,00 zł