O'REILLY®

Domain-Driven Design w transformacji systemów

Skuteczna modernizacja legacy bez zbędnego ryzyka

Carola Lilienthal
Henning Schwentner

Helion

Tytuł oryginału: Domain-Driven Transformation: Modernize Legacy Systems and Mitigate Risk

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-3881-6

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/dddtra
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *helion.pl* (księgarnia internetowa, katalog książek)

Printed in Poland.

Spis treści

Słowo wstępne **13**

Przedmowa **14**

1. Domain-Driven Transformation w pigułce **19**
Przykład: system zastany w porcie kontenerowym 19
Ponowne odkrywanie dziedziny 21
Określanie docelowej architektury 24
Porównanie architektury rzeczywistej z docelową 25
Określanie priorytetów i realizowanie transformacji 32
Co się właśnie stało? 33
Metoda 34
Ścieżki przez DDT 35
Podsumowanie 42

Część I. Podstawy **43**

2. Jak zapanować nad złożonością **45**
Źródła złożoności — przestrzeń problemu i przestrzeń rozwiązań 46
Rodzaje złożoności — zasadnicza a przypadkowa 47
Źródła złożoności przypadkowej 48
Obszary decyzyjne w architekturze oprogramowania 49
Radzenie sobie ze złożonością zasadniczą i przypadkową 50
Złożoność w systemach zastanych 51
Metody podstawowe 52
Co dalej? 53

3. Domain-Driven Design 54
Koncepcje DDD w walce ze złożonością 55
DDD a dziedzina biznesowa 56
Język wszechobecny 58
Projektowanie strategiczne po stronie biznesowej 64
Kategoryzowanie poddziedzin — co jest dziedziną główną? 66
DDD a rozwiązanie programowe 70
Projekt strategiczny w oprogramowaniu 70
Mapowanie kontekstów 72
Kontekst bańkowy 76
Model dziedzinowy 77
Architektura warstwowa w DDD 82
Projektowanie taktyczne 82
Zdarzenia dziedzinowe 88
Podsumowanie 90

4. Modelowanie kooperacyjne 91
Narzędzia do modelowania i monopolizacja modelu 92
Metody modelowania współpracy w transformacji 93
Łączenie właściwych osób 94
Podstawowe koncepcje modelowania kooperacyjnego 95
Modelowanie kooperacyjne i DDD 97
Domain Storytelling — opowiadanie historii dziedzinowych 98
Jeden obraz — jedna historia 101
Rysowanie granic 101
Czynniki zakresu w historiach dziedzinowych 103
Znajdowanie modelu dziedzinowego 105
Perspektywy 106
EventStorming 106
Eksploracja w formacie Big Picture 107
Eksploracja chaotyczna 107
Egzekwowanie osi czasu 109
Ludzie i systemy 110
Przejście krok po kroku 110
Działania dodające wartość 111
Rysowanie granic 111
Kodowanie kolorami 112
Więcej szczegółów — Software Design EventStorming 113
Znajdowanie modelu dziedzinowego 115
Inne formaty i perspektywy 116

Scenario Casting 116
Jak działa Scenario Casting? 116
Podstawowa idea 117
Przebieg warsztatu — od eksploracji do wdrożenia 117
Studium przypadku — Green Urban Supplier 118
Krok 1. Burza mózgów 119
Krok 2. Skupienie 122
Krok 3. Składanie 125
Co dalej w podejściu scenariuszowym? 127
Druga runda Scenario Castingu 130
Perspektywy 131
Którą metodę wybrać? 131
Zdalne modelowanie kooperacyjne 133
Modelowanie kooperacyjne a AI 133
Inne metody modelowania kooperacyjnego 133
Podsumowanie 134

5. Koncepcje architektury oprogramowania 135
Dokumentowanie architektury z wykorzystaniem modelu C4 136
Poziom 1. Kontekst 136
Poziom 2. Kontenery 137
Poziom 3. Komponenty 138
Poziom 4. Kod 139
Modularność 140
Spójność i sprzężenie 140
Spójność i sprzężenie w DDD 142
Spójność i sprzężenie w dziedzinie 142
Wielka bryła błota 143
Na poziomie klasy 143
Na poziomie architektury 144
Skąd bierze się wielka bryła błota? 146
Sprzężenie a wielokrotne używanie kodu 147
Style architektoniczne systemów zastanych 148
Monolit 148
Mikrousługi 149
Od monolitu do mikrousług 151
Stara dobra architektura warstwowa 152
Architektury wewnętrzno-zewnętrzne: heksagonalna (czyli porty i adaptery), cebulowa, czysta i inne 153
Połączenie różnych stylów — architektura jawna 154
Architektury wewnętrzno-zewnętrzne a taktyczne DDD 154

Hamburger architektury oprogramowania 156
Architektury zorientowane na usługi 158
Systemy samodzielne 162
Od krajobrazów IT do SOA i z powrotem 162
Indeks dojrzałości modularności — na jakim etapie jesteś? 164
Modularność w MMI 165
Hierarchiczność w MMI 166
Spójność wzorców w MMI 166
Obliczanie MMI 167
Podsumowanie 168

6. Podejścia do transformacji ... 170
Wymiana — wielki wybuch 171
Wymiana — krok po kroku 174
Przekształcanie 176
Podsumowanie 177

Część II. Transformacja techniczna, taktyczna i zespołowo-organizacyjna 179

7. Techniczna stabilizacja oprogramowania zastanego ... 181
Aktualizacja zintegrowanego środowiska programistycznego 181
Automatyzacja budowania i wdrażania 182
Aktualizowanie zależności oprogramowania 183
Aktualizacja bibliotek i platform 183
Aktualizacja języka programowania 184
Eliminowanie ostrzeżeń kompilatora i środowiska IDE 185
Zwiększanie pokrycia testami przy naprawianiu błędów 185
Zasada skauta 186
Różne rodzaje testów 187
Porządkowanie kodu na bieżąco 188
Refaktoryzacja 189
Eliminowanie zależności 190
Poprawa wewnętrznej struktury kodu z użyciem miar 192
Zwiększanie odporności na błędy 193
Budowanie odporności na wartości null 193
Poprawa użycia wyjątków 194
Wprowadzenie do projektowania kontraktowego 196
Podsumowanie 197
Na jakim jesteś etapie? — łatwa wersja 198

8. Wzmacnianie kodu źródłowego wiedzą dziedzinową 199
Oddzielanie kodu biznesowego od technicznego 200
Zwiększanie spójności 201
Wzbogacanie modeli wiedzą dziedzinową 202
Używanie obiektów wartości 205
Wprowadzanie projektowania kontraktowego 208
Jawne określanie tożsamości encji 211
Ograniczanie sprzężenia 212
Eliminowanie cykli i zależności 212
Wprowadzanie referencji tożsamościowych na granicach agregatów 213
Ograniczanie dziedziczenia w biznesowym kodzie źródłowym 215
Zasada podstawienia Liskov 215
Dokumentowanie architektury w kodzie 218
Czy AI ma wiedzę o mojej dziedzinie? 220
Podsumowanie 220
Na jakim jesteś etapie? — łatwa wersja 220

9. Usprawnianie organizacji pracy zespołu 222
Oprogramowanie jako system socjotechniczny 222
Architektura i organizacja horyzontalna 223
Architektura i organizacja wertykalna 225
Jak przeprowadzić reorganizację zespołu? 226
Topologie zespołów 227
Praca w stanie przepływu 227
Typy topologii 228
Tryby interakcji między zespołami 231
Ewolucja zespołów w czasie 233
Zespoły wspomagające modernizację architektury 234
Podsumowanie 237
Na jakim jesteś etapie? — łatwa wersja 237

Część III. Strategiczna transformacja dziedzinowa 239

10. Krok 1. Ponowne odkrywanie dziedziny 243
Schematyczny opis kroku 1. 244
Wyzwanie: zbieranie odpowiednich ekspertów dziedzinowych 246
Wyzwanie: kierowanie ekspertów dziedzinowych z powrotem do właściwej dziedziny 247
Podkrok 1.1. Zrozumienie stanu obecnego 248
Podkrok 1.2. Wyodrębnianie wiedzy dziedzinowej (oczyszczanie) 248
Technika — eksperyment myślowy „proces na papierze” 249
Technika — eksperyment myślowy „wirtualni aktorzy” 249

Alternatywa: modelowanie na czysto od samego początku 250
Destylowanie języka wszechobecnego 251
Podkrok 1.3. Optymalizacja procesów biznesowych 252
Podkrok 1.4. Znajdowanie poddziedzin w procesach biznesowych 252
Wskaźniki granic poddziedzin 253
Wyzwanie: unikanie dekompozycji według elementów roboczych 258
Wyzwanie: znajdowanie odpowiedniego rozmiaru 261
Cele doboru odpowiedniego rozmiaru 261
Wskaźniki odpowiedniego rozmiaru 262
Dyskutowanie o dekompozycji z AI 262
Podsumowanie 263

11. Krok 2. Modelowanie architektury docelowej 264
Schematyczny opis kroku 2. 264
Podkrok 2.1. Tworzenie mapy kontekstowej 266
Wyzwanie: konteksty wynikające z przestrzeni rozwiązań 268
Podkrok 2.2. Klasyfikowanie poddziedzin 270
Podkrok 2.3. Przypisywanie kontekstów ograniczonych do zespołów 273
Wyzwanie: przydział zespołów to często kształtowanie zespołów 273
Podkrok 2.4. Określanie relacji 274
Składanie wszystkiego w całość 276
Podsumowanie 277

12. Krok 3. Dopasowanie architektury rzeczywistej do docelowej 278
Schematyczny opis kroku 3. 280
Podkrok 3.1. Identyfikowanie rzeczywistej architektury 281
Rzeczywista architektura a sztuczna inteligencja 282
Podkrok 3.2. Porównywanie docelowej mapy kontekstowej z kodem źródłowym 284
Wyzwanie: modele nieograniczone 286
Dekompozycja od wewnątrz na zewnątrz 286
Wyzwanie: nieograniczony model dziedzinowy 288
Przyczyna problemu: błędne rozumienie wielokrotnego wykorzystywania kodu 288
Wyzwanie: nieograniczony anemiczny model dziedzinowy 291
Wyzwanie: nieograniczony model danych 297
Wyzwanie: dekompozycja interfejsu użytkownika 298
Monolityczny interfejs użytkownika 299
Interfejs użytkownika w kontekście — mikrofrontendy 299
Podkrok 3.3. Tworzenie listy refaktoryzacji i ustaleń 302
Podsumowanie 304

13. Krok 4. Określanie priorytetów i realizacja działań transformacyjnych 305
Schematyczny opis kroku 4. 306
Podkrok 4.1. Wybór pierwszej refaktoryzacji strategicznej 307
Wzorzec: zaczynanie od poddziedziny pomocniczej 308
Wzorzec: przejście do prac nad dziedziną główną 308
Podkrok 4.2. Dodawanie refaktoryzacji taktycznych do backlogu produktu (lub backlogu usprawnień) 309
Podkrok 4.3. Planowanie wykonalnych refaktoryzacji w backlogu sprintu 310
Przerywanie refaktoryzacji tylko w sytuacjach awaryjnych 310
Szacowanie refaktoryzacji 311
Domain-Driven Transformation z użyciem metodyki Kanban 311
Podkrok 4.4. Wyodrębnianie kontekstu 313
Wyzwanie: rozplątywanie logiki biznesowej w serwisach 313
Wyzwanie: przenoszenie lokalnych danych, metod ustawiających i pobierających do kontekstu 318
Tożsamość encji 321
Wyzwanie: duplikowanie danych, metod ustawiających i pobierających do wielu kontekstów 323
Wyzwanie: wprowadzanie zdarzeń dziedzinowych między usługami 327
Wyzwanie: przenoszenie logiki dziedzinowej z usług do anemicznego modelu dziedziny 329
Przenoszenie logiki z usług do anemicznych encji 329
Wyzwanie: rozplątywanie modelu danych 333
Wyzwanie: wybieranie ustaleń w odpowiednim momencie 336
Wyzwanie: utrzymywanie motywacji 337
Podsumowanie 338

Część IV. Podsumowanie 339

14. Perspektywy — wzorce dziedzinowe i ich implementacja w kontekstach ograniczonych 341
Wymiar 1. Przepływ pracy w dziedzinie 342
Wzorzec dziedzinowy Potok 342
Wzorzec dziedzinowy Tablica 343
Wzorzec dziedzinowy Dialog 344
Ewolucja dziedzin i wzorce ich rozwoju 345
Wymiar 2. Model dziedzinowy w oprogramowaniu 345
Centralny element dziedzinowy a zróżnicowane elementy dziedzinowe 346
Formularzowy model dziedzinowy 348

Wymiar 3. Kształt dziedziny 349
Wiek i dojrzałość dziedziny 349
Dziedzina obiektów materialnych a dziedzina w pełni cyfrowa 350
Przestrzeń na transformację 350
Podsumowanie 351

15. Podsumowanie metody Domain-Driven Transformation ... 352

A Przegląd katalogu refaktoryzacji dziedzinowych ... 355

B Refaktoryzacje strategiczne ... 356

C Refaktoryzacje taktyczne wspierające refaktoryzacje strategiczne ... 360

D Refaktoryzacje socjotechniczne ... 366

E Refaktoryzacje taktyczne wzmacniające wiedzę dziedzinową ... 371

Bibliografia ... 382

Skorowidz ... 389

ROZDZIAŁ 1.

Domain-Driven Transformation w pigułce

Kiedy rozpoczynasz nowy projekt oprogramowania, wszystko jest przyjemne i łatwe. Programiści błyskawicznie reagują na nowe wymagania. Użytkownicy są pełni entuzjazmu. Prace posuwają się naprzód wielkimi krokami.

W czasie życia systemu sytuacja ta nieuchronnie się zmienia. Złożoność implementacji oprogramowania rośnie. Rosnąca złożoność prowadzi do większej podatności na błędy, coraz wolniejszych postępów i trudniejszej konserwacji. W końcu nawet najmniejsza zmiana potrzebuje pół roku, żeby trafić na produkcję. Kwitnąca zielona łąka zamienia się w błotniste, cuchnące pole. „System zastany", „starożytne oprogramowanie", „wielka bryła błota", „monolit" i „projekt bagno" to niepochlebne określenia nadawane tego typu systemom.

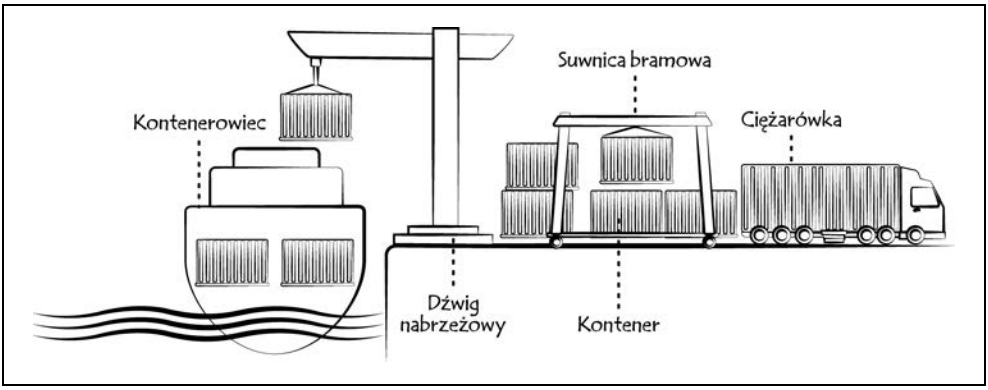
Jest jednak nadzieja! Nawet te starzejące się systemy mogą odzyskać elastyczność, solidność i szybkość rozwoju. W tym rozdziale najpierw przedstawimy konkretny przykład systemu zastanego, który jest modernizowany z użyciem Domain-Driven Transformation, a następnie opiszemy tę metodę w zarysie. Pozwoli Ci to zarówno zrozumieć ogólne podejście, jak i zorientować się, które z wielu tematów poruszanych w książce są dla Ciebie szczególnie interesujące i które rozdziały warto przeczytać.

Przykład: system zastany w porcie kontenerowym

Pradziadek Henninga był handlarzem kakao. Ze swojego biura w porcie obserwował ogromne parowce oceaniczne wpływające do portu i wypływające. Gdy otwierał okno, mógł poczuć zapach wielkiego świata — dojrzewających bananów, palonej kawy, a czasem nawet dzikich zwierząt. W tamtych czasach, około 1930 r., statki były rozładowywane przez dokerów — mężczyzn, którzy przenosili ładunek w workach lub skrzyniach na plecach ze statków do wielkich magazynów.

Po wynalezieniu kontenera w latach 60. zeszłego wieku praca w porcie diametralnie się zmieniła. Magazyny zostały opuszczone i zastąpione terminalami kontenerowymi. Dokerzy zostali przeszkoleni do obsługi pojazdów, które teraz służyły do transportu kontenerów i ładunków na terenie terminalu.

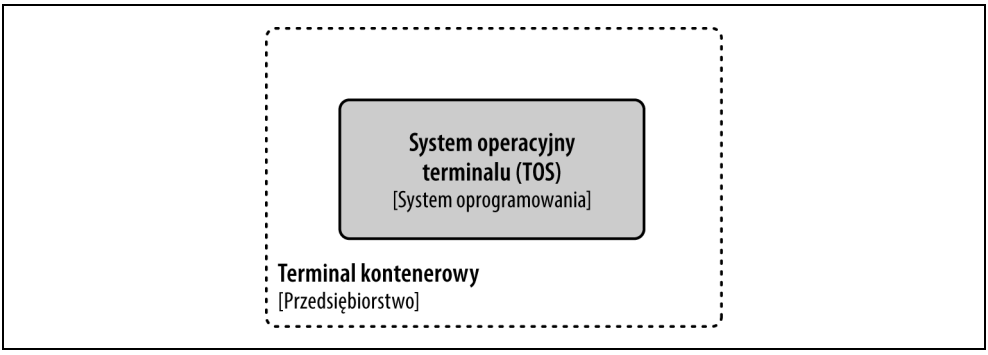
Na rysunku 1.1 pokazano szkic poszczególnych części terminalu kontenerowego. Kontenery są rozładowywane ze statku za pomocą dźwigu nabrzeżowego, transportowane z nabrzeża na plac składowy, a następnie przenoszone na ciężarówkę lub pociąg towarowy przy użyciu suwnicy bramowej. Kontenery dostarczane do terminalu ciężarówką lub pociągiem towarowym w celu transportu statkiem pokonują odwrotną trasę.



Rysunek 1.1. Terminal kontenerowy — schemat

W latach 70. i 80. terminale kontenerowe zaczęły wykorzystywać oprogramowanie do kontrolowania gromadzonych danych i stopniowej automatyzacji procesów. Obecnie terminal kontenerowy dysponuje infrastrukturą IT obejmującą różnorodne systemy — do planowania załadunku i rozładunku kontenerowców, do organizacji przyjmowania i wydawania kontenerów przy bramie dla ciężarówek, a także do zarządzania placem składowym i organizacji transportu na jego terenie.

Często kilka z tych funkcji jest zaimplementowanych w jednym systemie, do którego podłączone są inne systemy peryferyjne. Ten centralny system nazywany jest systemem operacyjnym terminalu (ang. *Terminal Operating System*, TOS). Dla uproszczenia w poniższym przykładzie będziemy mówić wyłącznie o TOS-ie, pomijając inne możliwe systemy (patrz rysunek 1.2).



Rysunek 1.2. System operacyjny terminalu

W terminalu kontenerowym działa zatem TOS, który przez 30 lat sumiennie wykonywał swoje zadania. Przez lata dobrze służył terminalowi i był wielokrotnie rozbudowywany oraz modernizowany.

Niestety te rozbudowy i modernizacje często były prowadzone chaotycznie, bez odpowiedniego planowania. W efekcie TOS rozrastał się w niekontrolowany sposób we wszystkich kierunkach. Obecnie TOS stał się podatnym na błędy, trudnym do rozszerzania, przestarzałym i monolitycznym systemem zastanym.

Właśnie dlatego dyrektor terminalu zatrudnił nas — Carolę i Henninga — jako konsultantów do modernizacji systemu TOS. Po przybyciu na miejsce z rysunku 1.1 i diagramu z rysunku 1.3 dowiadujemy się, że z systemu TOS korzystają kontrolerzy bramowi, operatorzy suwnic, operatorzy dźwigów, dyspozytorzy oraz planiści sztauowania[1,2]. Każda z tych ról ma w TOS-ie własne ekrany z funkcjonalnościami dostosowanymi do konkretnych zadań. Operatorzy dźwigów i wózków bramowych mają również dedykowaną aplikację TOS na urządzeniach mobilnych w kabinach operatorskich.

Notacja (nazywana modelem C4) użyta na rysunkach 1.2 i 1.3 została wyjaśniona w rozdziale 5. Na razie wystarczy zapoznać się z ramką „Pomocnik transformacyjny — model C4".

Pomocnik transformacyjny — model C4

Model C4 to hierarchiczne podejście do architektury oprogramowania, które wykorzystuje cztery typy diagramów do przedstawienia systemu na różnych poziomach abstrakcji, żeby interesariusze mogli lepiej go zrozumieć. W tym przypadku używamy widoku najwyższego poziomu — diagramu kontekstowego, który przedstawia użytkowników, systemy zewnętrzne oraz ich wzajemne interakcje, aby pokazać, jak system wpisuje się w swoje otoczenie.

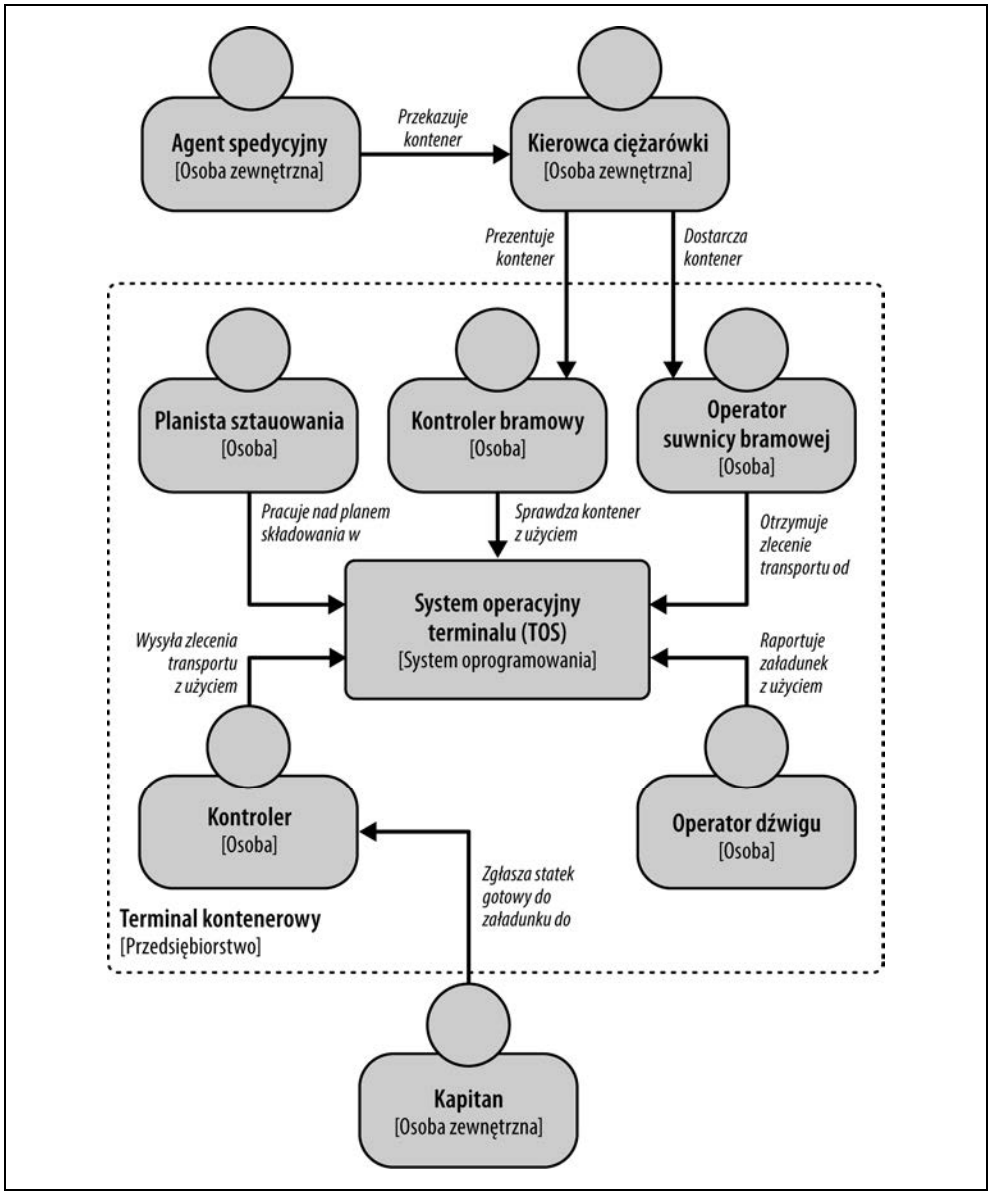
Ponowne odkrywanie dziedziny

Teraz gdy wiemy, kto pracuje ze starzejącym się TOS-em, zapraszamy wybranych użytkowników systemu (czyli pracowników z różnych obszarów terminalu kontenerowego) na warsztaty. Przeprowadzamy z nimi wywiady i pytamy, jak przebiega ich praca na ogólnym poziomie.

Ten krok zaskakuje architektów i programistów starzejącego się systemu. Spodziewali się, że najpierw przeanalizujemy kod źródłowy systemu, aby znaleźć miejsca wymagające modernizacji. Takie podejście z pewnością może prowadzić do poprawy jakości, ale nie rozwiązuje podstawowego problemu systemów przestarzałych. Architektura systemu nie opiera się na różnych

[1] Uwaga: transformacja ma sens tylko w przypadku złożonych systemów. Aby zapewnić Ci przegląd metody, ten wstępny przykład został uproszczony na wiele różnych sposobów — dziedzina jest mniej skomplikowania, architektura mniej splątana, a kroki do podjęcia prostsze niż w rzeczywistym świecie. Pomimo to mamy nadzieję, że przykład ten da Ci ogólne pojęcie o metodzie i zachęci do sięgnięcia głębiej. Zagadnienia, o których tu jedynie wspominamy, zostaną wyjaśnione obszerniej i dokładniej w kolejnych częściach książki.

[2] W prawdziwym terminalu kontenerowym są inne role, które pomijamy tu dla prostoty.



Rysunek 1.3. TOS w kontekście

obszarach biznesowych, które można rozpoznać w dziedzinie, lecz jest wynikiem niekontrolowanego rozrostu. Obszary biznesowe można odnaleźć tylko wtedy, *gdy dziedzina biznesowa zostanie odkryta na nowo* — dlatego zaczynamy od rozmów z pracownikami.

Pracownicy opowiadają nam o typowej podróży kontenera przez terminal:

1. Zanim nastąpi jakikolwiek transport, agent spedycyjny wysyła dokumenty dostawy kontenera do terminalu kontenerowego — dzięki temu terminal może zaplanować działania z wyprzedzeniem.
2. Gdy towary zostaną zapakowane do kontenera i wszystkie dokumenty będą gotowe u agenta spedycyjnego, ciężarówka z kontenerem jedzie do bramy terminalu kontenerowego.
3. Kontrolerzy bramowi sprawdzają dokumenty dostawy i ładunek. Jeśli wszystko jest w porządku, ciężarówka z kontenerem może wjechać do punktu przeładunkowego na terminalu. Stamtąd suwnica bramowa zabiera kontener i umieszcza go na placu składowym.
4. Kilka dni przed planowanym rozpoczęciem podróży kontenera statkiem planista sztauowania umieszcza kontener w planie załadunku statku, który ma go przewieźć. Po przybyciu statku do terminalu kontenerowego kapitan zgłasza się do kontrolera odpowiedzialnego za statek i można rozpocząć załadunek.
5. Kontroler wysyła operatorowi suwnicy bramowej zlecenie transportu. Suwnica zabiera kontener i przewozi go na nabrzeże przy miejscu cumowania statku. Tam dźwig ładuje kontener do odpowiedniego przedziału na statku.
6. Na koniec statek odpływa i ostatecznie dociera do portu docelowego, gdzie kontener zostaje rozładowany.

Słuchając tej opowieści, zapisujemy ją jako historię dziedzinową (patrz rysunek 1.4)[3]. Ta historia dziedzinowa zawiera różne role z rysunku 1.3 i pokazuje, jak przebiega praca bez uwidaczniania samego systemu oprogramowania.

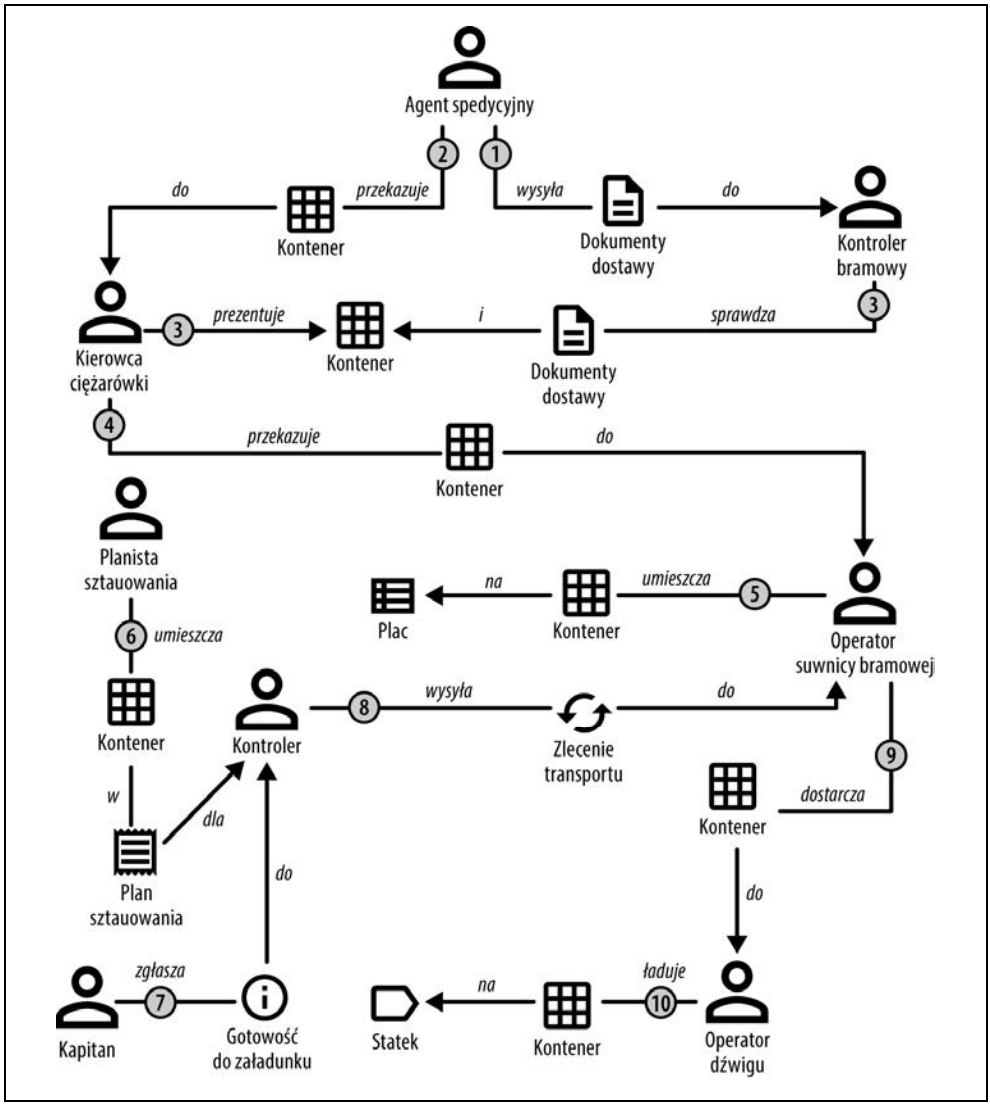
Wspólne tworzenie historii dziedzinowych pozwala nam — zarówno konsultantom, jak i ekspertom dziedzinowym — dużo się dowiedzieć. Na przykład operator suwnicy nie wiedział, w jaki sposób przebiega weryfikacja dokumentu dostawy. Teraz wie, dlaczego czasem musi tak długo czekać na przekazanie kontenera.

Sposób tworzenia historii dziedzinowych podobnych do tej z rysunku 1.4 oraz stosowana w nich notacja zostaną wyjaśnione w rozdziale 4. Aby zrozumieć tyle, ile potrzeba na tę chwilę, zajrzyj do ramki „Pomocnik transformacyjny — opowiadanie historii dziedzinowych".

Pomocnik transformacyjny — opowiadanie historii dziedzinowych

Opowiadanie historii dziedzinowych (ang. *Domain Storytelling*) to metoda modelowania zespołowego, która łączy ekspertów dziedzinowych z deweloperami w celu wspólnego zrozumienia działalności firmy. Uzyskaną wiedzę wizualizuje się na diagramie przedstawiającym aktorów (ludziki), obiekty pracy (pozostałe ikony) oraz czynności (strzałki). Aby odczytać historię we właściwej kolejności, należy podążać za numerami i strzałkami.

[3] Dla uproszczenia pominęliśmy niektóre szczegóły w tej historii dziedzinowej.



Rysunek 1.4. Ogólny schemat procesu w terminalu kontenerowym. Podążaj za numerami i strzałkami, aby przeczytać historię we właściwej kolejności

Określanie docelowej architektury

Gdy wszyscy zrozumieli ogólny proces i z pewnością poznali wiele szczegółów dotyczących pracy w terminalu kontenerowym, kolejnym zadaniem jest zidentyfikowanie obszarów biznesowych (zwanych poddziedzinami) w tym ogólnym procesie.

Następuje intensywna dyskusja z ekspertami dziedzinowymi. Raz po raz rozważamy następujące kwestie:

- Które kroki lub czynności w historii dziedzinowej są ze sobą powiązane?
- Gdzie możemy zidentyfikować spójne obszary biznesowe, które tworzą poddziedzinę?
- Jakie są właściwe nazwy dla poddziedzin?

Dowiadujemy się na przykład, że agent spedycyjny, kierowca ciężarówki i kontroler bramowy współpracują ze sobą w obszarze biznesowym, którego celem jest wprowadzenie kontenera do portu. Pracownicy terminalu kontenerowego nazywają ten obszar biznesowy „wjazdem do portu". Jednym ze wskaźników pomagających znaleźć granicę obszaru jest jednokierunkowy charakter kroku 5. Po tym kroku nie ma powrotu do wjazdu do portu ani do aktorów — agenta spedycyjnego, kontrolera bramowego i kierowcy ciężarówki.

W wyniku odbytych rozmów rysujemy kilka ramek w historii dziedzinowej, aby pogrupować powiązane kroki w poddziedziny (patrz rysunek 1.5).

Sposób znajdowania poddziedzin oraz heurystyki i wskaźniki, które mogą w tym pomóc, zostały wyjaśnione w rozdziałach 3. i 10.

Następnie omawiamy poddziedziny, które są obsługiwane w systemie. Nazwy tych poddziedzin zapisujemy na tablicy i dodajemy zależności między poszczególnymi częściami procesu. W rezultacie powstaje tak zwana mapa kontekstowa z zależnymi od dziedziny kontekstami ograniczonymi (patrz rysunek 1.6). Ta mapa kontekstowa stanowi szablon naszej docelowej architektury.

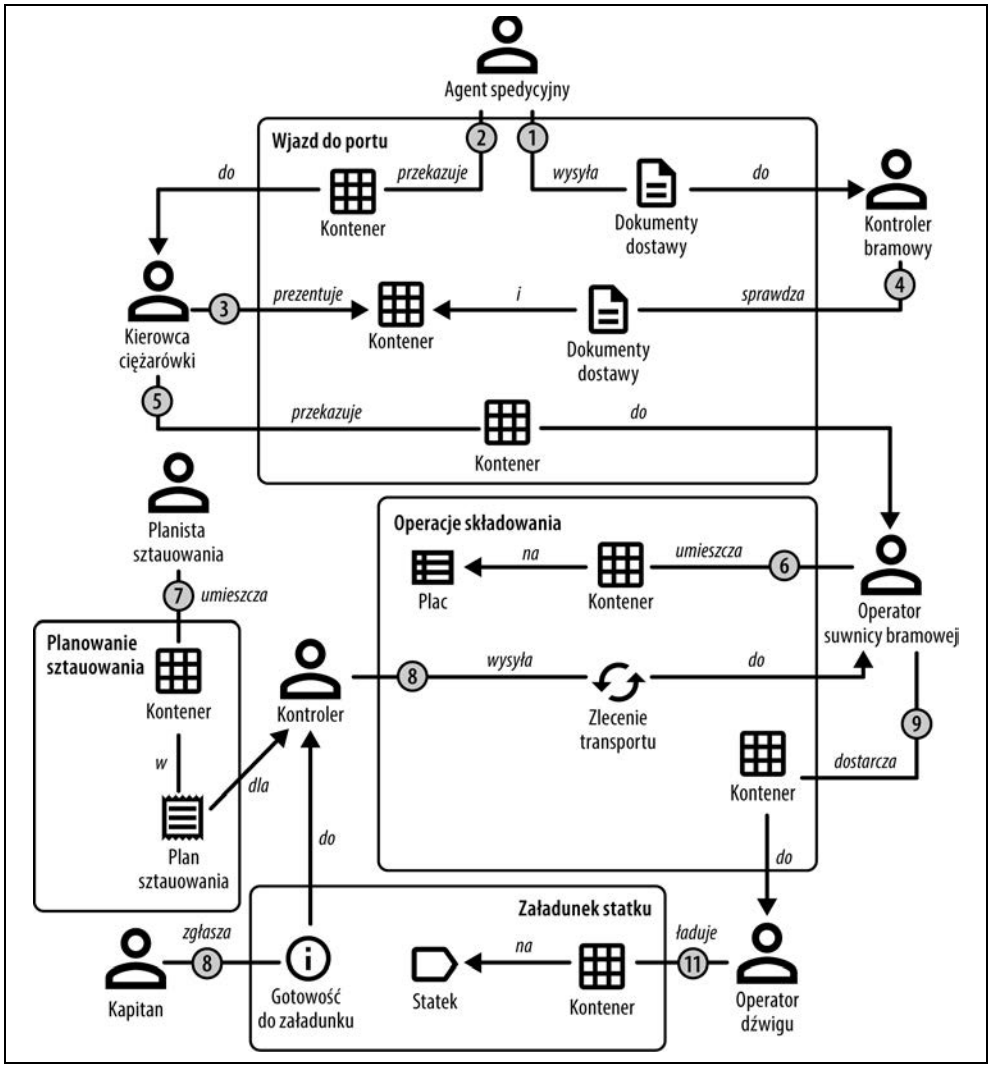
Dla każdego ograniczonego kontekstu z mapy kontekstowej powinien istnieć odpowiadający mu dziedzinowy moduł w docelowej architekturze, który wymienia informacje z innymi modułami poprzez wąskie interfejsy, jak na mapie kontekstu.

Sposób wyprowadzania ograniczonych kontekstów z poddziedzin oraz tworzenie i wykorzystywanie map kontekstowych opisano w rozdziałach 3. i 11.

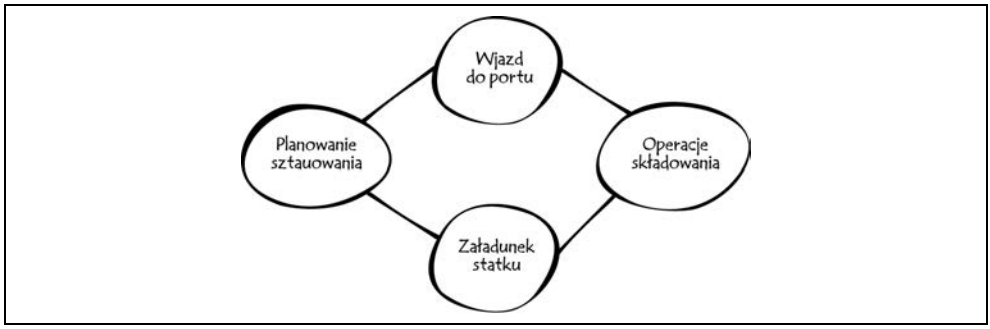
Porównanie architektury rzeczywistej z docelową

Skoro wiemy już, jak *powinien* wyglądać nasz system, możemy porównać architekturę docelową z rzeczywistą. Na tym etapie architekci i programiści są zadowoleni, bo wreszcie zajmujemy się ich kodem źródłowym. TOS został napisany w Javie. Aby porównać architekturę docelową z rzeczywistą, wczytujemy kod źródłowy TOS-a do narzędzia do przeglądu architektury i modelujemy pożądaną strukturę w kodzie. Informacje o tym, które narzędzia najlepiej nadają się do tego kroku, można znaleźć w książce *Sustainable Software Architecture* (Lilienthal 2019).

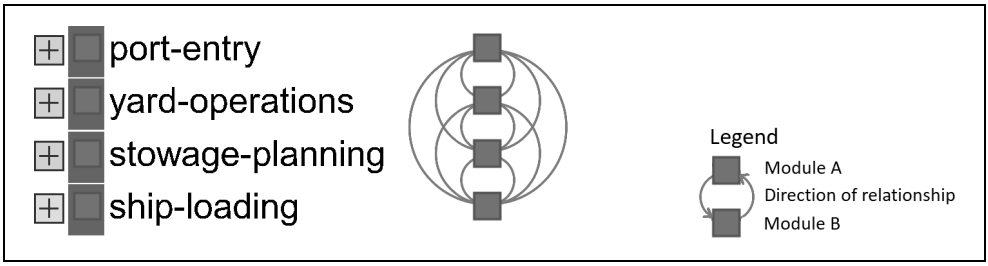
Na rysunku 1.7 widać nieciekawe wyniki tego pierwszego porównania. Na diagramie pokazano cztery konteksty ograniczone z mapy kontekstowej przedstawione jako kwadraty. Będziemy je nazywać *modułami*. Utworzyliśmy te moduły w narzędziu do przeglądu architektury i przypisaliśmy do nich odpowiednie części drzewa pakietów wraz z zawartymi w nich klasami.



Rysunek 1.5. Ogólny proces terminalu kontenerowego — z zaznaczonymi granicami



Rysunek 1.6. Docelowa mapa kontekstowa terminalu kontenerowego



Rysunek 1.7. Zrzut ekranu narzędzia do przeglądu architektury; porównanie architektury aktualnej i docelowej. Jeśli nie znasz narzędzi tego typu, zajrzyj do ramki „Pomocnik transformacyjny — narzędzia do przeglądu architektury"

Pomocnik transformacyjny — narzędzia do przeglądu architektury

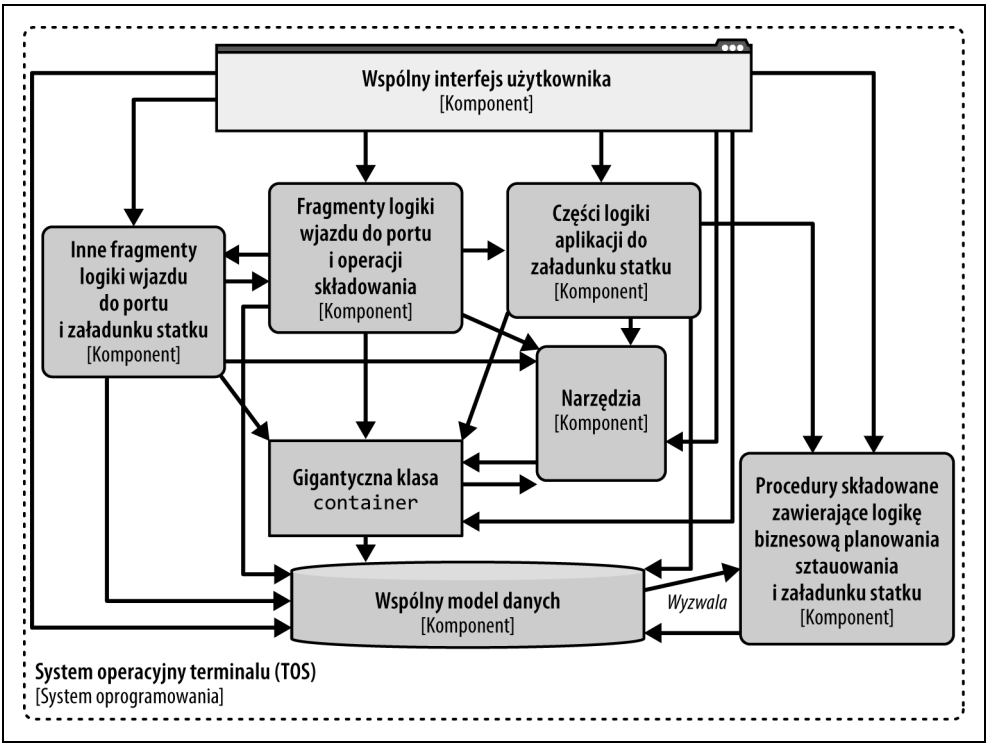
Narzędzie do przeglądu architektury może pomóc w wizualizacji i zrozumieniu aktualnej architektury. Przydaje się też do porównywania jej z architekturą docelową.

W tej książce pokazujemy zrzuty ekranu z narzędzia o nazwie Sonargraph. Po kliknięciu znaku plus przed kwadratami (modułami) rozwija się drzewo pakietów i klas. Po prawej stronie diagramu widoczne są rzeczywiste relacje z kodu źródłowego, przedstawione jako łuki. Łuki znajdują się również po lewej stronie. Reprezentują one relacje między klasami posortowanymi w moduły. Łuki po lewej stronie biegną od góry do dołu, a po prawej stronie — od dołu do góry.

Rysujemy odpowiedni diagram komponentów na tablicy (ponownie w notacji modelu C4; patrz rysunek 1.8), aby wszyscy programiści i architekci mogli zrozumieć wyniki analizy z narzędzia pokazanego na rysunku 1.7. Wskazują one, że w systemie powinny istnieć z jednej strony cztery moduły dziedzinowe, a z drugiej strony — struktura techniczna składająca się z warstw interfejsu użytkownika, aplikacji dla procesów dziedzinowych, modelu dziedzinowego oraz bazy danych. Tak powinna wyglądać dobrze ustrukturyzowana architektura modularna, *ale obecnie TOS w żaden sposób nie odzwierciedla tej koncepcji architektonicznej!* Mamy do czynienia z silnie powiązaną strukturą z wieloma cyklami zależności.

Strzałki między ramkami na rysunku 1.8 wskazują, że między komponentami systemu istnieje wiele zależności; niektóre z nich są nawet cykliczne. Niezachęcający rezultat to silnie powiązane komponenty, które w jakiś sposób zapewniają funkcjonalność dla czterech poddziedzin. Widać przynajmniej pewne *warstwy techniczne* — wspólny interfejs użytkownika tworzy warstwę górną, wspólna baza danych tworzy warstwę dolną, a wszystko inne (logika biznesowa) tworzy warstwę środkową.

Wielokrotnie widzieliśmy systemy w takim stanie i nie dziwi nas już, że TOS jest podatny na błędy i trudny do rozbudowy. Aby zrozumieć prawdziwą skalę problemu, określamy indeks dojrzałości modularności (ang. *Modularity Maturity Index*, MMI) tego systemu.

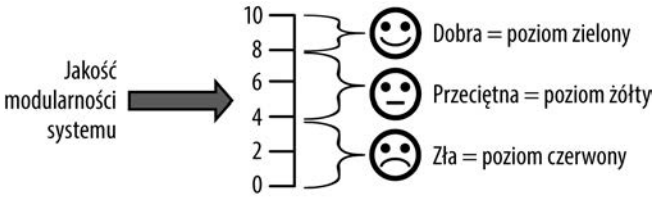


Rysunek 1.8. TOS — obecnie silnie powiązana struktura z wieloma cyklami zależności, ale także z pewnym warstwowym podziałem technicznym

Dokładne informacje o tym, jakie aspekty ocenia MMI oraz jak się go wyznacza i oblicza, znajdziesz w rozdziale 5. Aby zrozumieć MMI w stopniu wystarczającym na tym etapie, zajrzyj do ramki „Pomocnik transformacyjny — indeks dojrzałości modularności".

Pomocnik transformacyjny — indeks dojrzałości modularności

MMI jest obliczany na podstawie szeregu miar i kryteriów jakościowych, dzięki czemu odwzorowuje jakość modularności systemu w skali od 0 do 10 (patrz rysunek 1.9).



Rysunek 1.9. Wskaźnik dojrzałości modularności

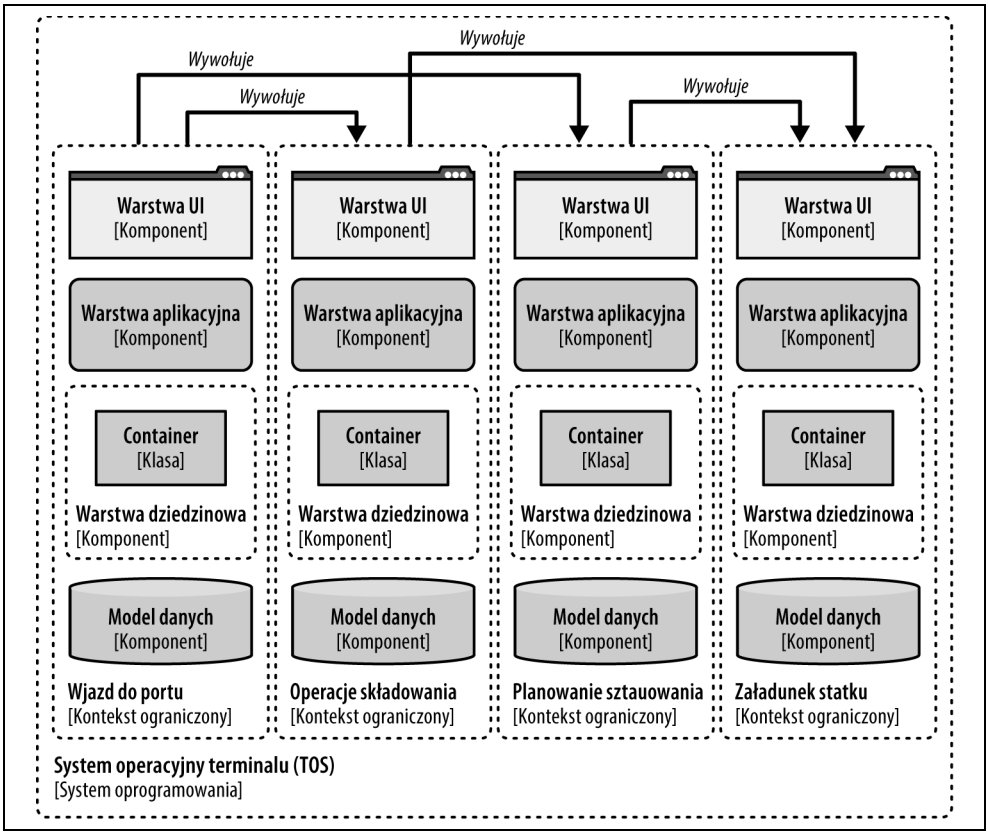
Obliczona wartość jest następnie podsumowywana w formie świateł drogowych. Dobrze ustrukturyzowane systemy mają MMI od 8 do 10, mierne mieszczą się w przedziale od 4 do 8, a systemy o złej strukturze mają MMI poniżej 4.

Wskaźnik MMI obliczony dla TOS-u wynosi 4,9 — system najwyraźniej znajduje się w przeciętnym stanie na wszystkich poziomach.

Wartość MMI 4,9 dla TOS-u wynika częściowo z faktu, że dziedzinowa koncepcja „kontenera" wraz ze wszystkimi funkcjami wjazdu do portu, operacji składowych, planowania sztauowania i załadunku statku została wymodelowana w jednej ogromnej klasie z kilkoma innymi klasami, do których delegowane są podzadania. Dane z klasy `Container` i powiązanych z nią elementów są z kolei przechowywane w jednej gigantycznej tabeli bazy danych.

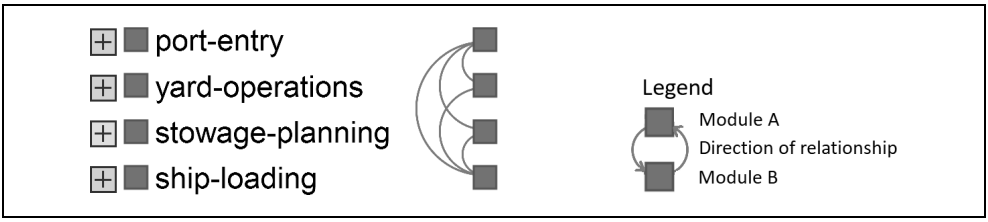
Na początku rozwoju TOS-u klasa `Container` i powiązana z nią tabela bazy danych były jeszcze małe i dobrze dopasowane do pierwszej części systemu, która została zaprogramowana jako mały TOS wspierający podstawowe procesy w terminalu kontenerowym. Jednak z każdym rozszerzeniem TOS o nową funkcję klasa `Container` i jej tabela w bazie danych stawały się coraz większe, tak że obecnie musimy mówić o *klasie-bogu* lub *nieograniczonym modelu dziedzinowym*.

Chcemy osiągnąć strukturę taką jak na rysunku 1.10, w której poszczególne konteksty ograniczone z rysunku 1.6 mają własne klasy `Container` zredukowane do odpowiedniego kontekstu i komunikują się ze sobą poprzez wąskie interfejsy.



Rysunek 1.10. Architektura docelowa — konteksty ograniczone z różnymi implementacjami klasy Container

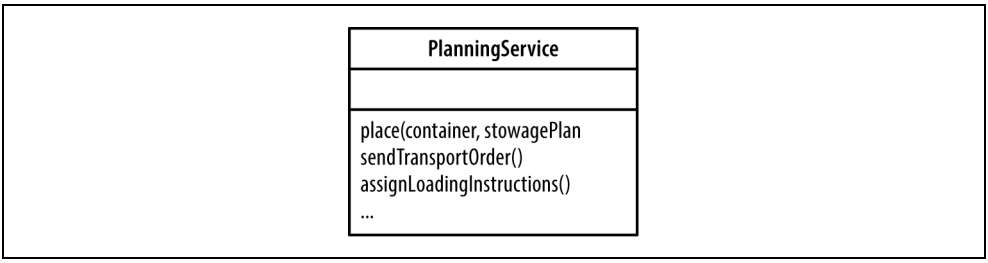
Struktura ta wyglądałaby również znacznie przejrzyściej w narzędziu do przeglądu architektury (patrz rysunek 1.11).



Rysunek 1.11. Zrzut ekranu narzędzia do przeglądu architektury pokazujący pożądany wynik dopasowania. Nie ma już żadnych relacji prowadzących w górę (które widniałyby po prawej stronie pól)

Aby uporządkować struktury i relacje w takim przypadku, potrzebny jest przemyślany plan, który podzieli ścieżkę od architektury aktualnej do docelowej na wykonalne etapy. W części III poznasz techniki, które pozwolą Ci samodzielnie opracować taki plan restrukturyzacji systemu zastanego (patrz rozdział 12.).

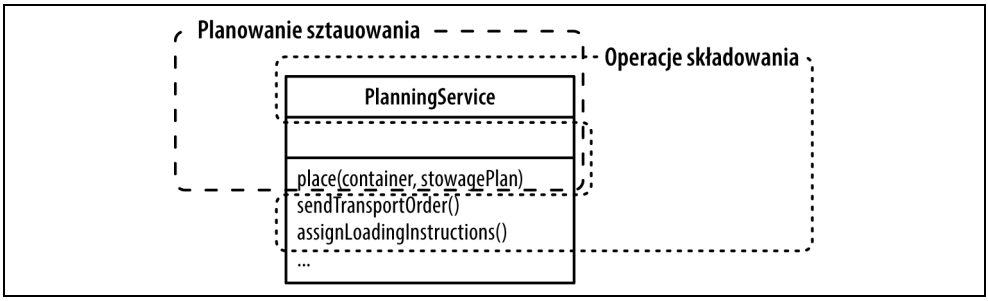
Punkt wyjścia dla naszego TOS-u zidentyfikowaliśmy podczas określania indeksu MMI (patrz wyżej), który omawiamy z programistami i architektami TOS-u, wypracowując w ten sposób część przyszłego planu konwersji. Naszym punktem wyjścia są wzorce projektowe, których użyli programiści TOS-u, aby wprowadzić pewien porządek w kodzie źródłowym. W kodzie znajdujemy szereg klas usługowych, z których wszystkie stosują wspólny wzorzec projektowy *Usługa* i oferują funkcjonalność bezstanową. Przykładem jest klasa `PlanningService`, której interfejs pokazano na diagramie klas na rysunku 1.12.



Rysunek 1.12. Rzeczywista architektura (wycinek). Problem: `PlanningService` łączy nieograniczoną logikę biznesową w jednej ogromnej klasie usługowej

Po bliższym zbadaniu interfejsu wspólnie z architektami i programistami zauważamy, że klasa ta obejmuje funkcje dla dwóch kontekstów ograniczonych z mapy kontekstowej (pokazanej wcześniej na rysunku 1.6): dla planowania sztauowania oraz operacji składowania. Na rysunku 1.13 nałożyliśmy oba konteksty na diagram klas `PlanningService`, aby lepiej uwidocznić problem.

Architekci i deweloperzy TOS-u przeszli wcześniej z nami szkolenie z Domain-Driven Design (DDD). Teraz przypominają sobie o projektowaniu strategicznym w DDD i zaczynają rozumieć, do czego dążymy (jeśli DDD jest dla Ciebie nowością, zajrzyj do ramki „Pomocnik transformacyjny — Domain-Driven Design”).



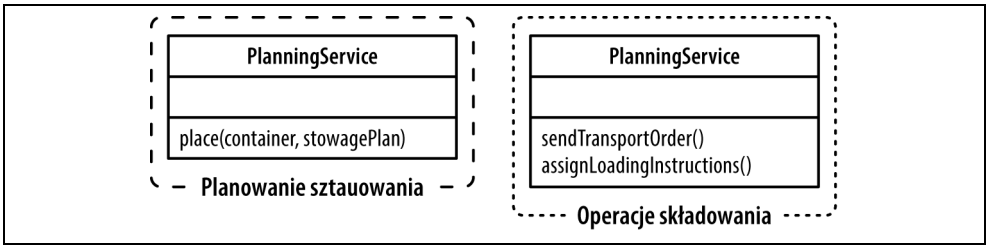
Rysunek 1.13. Dopasowanie architektury rzeczywistej do docelowej dla usługi `PlanningService`. Problem: w oprogramowaniu nie ma jeszcze wyrażonych granic; obszary „planowania sztauowania" (linia przerywana) i „operacji składowania" (linia kropkowana) nakładają się na siebie (co utrudnia odczytanie diagramu)

Pomocnik transformacyjny — Domain-Driven Design

Projektowanie dziedzinowe (ang. *Domain-Driven Design*, DDD) to filozofia tworzenia oprogramowania, w której zrozumienie biznesu i jego procesów determinuje architekturę systemu. Procesy, struktura, elementy pracy oraz język dziedziny są modelowane w kodzie.

DDD dostarcza kilka narzędzi; jedno z nich — projektowanie strategiczne — pomaga znaleźć granice w dziedzinie, aby wypracować właściwą modularyzację oprogramowania.

Architekci i programiści rozpoczynają dyskusję o tym, jak można podzielić obecną klasę `PlanningService`. Celem jest utworzenie dwóch oddzielnych klas `PlanningService`, z których każda musi istnieć w przeznaczonym dla niej module dziedzinowym. Podczas rozważania implementacji poszczególnych metod pojawiają się trudności i dyskusje, ale programiści i architekci wypracowują również pomysły na to, jak przeprowadzić pomyślnie separację klasy `Planning ↪Service`. Docelową architekturę z dwiema oddzielnymi klasami przedstawiono na rysunku 1.14.



Rysunek 1.14. Architektura docelowa na poziomie klas (fragment). Usprawnienie: funkcje już się nie nakładają; „planowanie sztauowania" (linia przerywana) i „operacje składowania" (linia kropkowana) to teraz oddzielne konteksty ograniczone

Jak pokazano wcześniej w przypadku klas `Container`, czasami warto podzielić jedną dużą klasę na kilka klas o tej samej nazwie w różnych kontekstach. Zespół nie jest szczególnie zadowolony z nazwy `PlanningService`. W tym przypadku lepszym rozwiązaniem byłoby znalezienie różnych nazw dla klas w poszczególnych kontekstach.

Ponadto obie klasy `PlanningService` łączą logikę aplikacji z logiką dziedzinową, więc lepiej byłoby dodatkowo oddzielić warstwę aplikacji od modelu dziedzinowego. Ponieważ nie da się zrobić wszystkiego naraz, zespół decyduje się odłożyć rozwiązanie tych problemów na później.

W trakcie dyskusji z programistami i architektami krok po kroku identyfikujemy kilka refaktoryzacji, które należy przeprowadzić w systemie. Najpierw zbieramy je na karteczkach samoprzylepnych na tablicy (patrz tabela 1.1). Niektóre refaktoryzacje, takie jak 1.0 i 2.0, są zbyt duże, by zrealizować je w jednym sprincie. Dzielimy je na mniejsze refaktoryzacje, takie jak 1.1, 1.2 i 2.1. Ważne jest, aby refaktoryzacje były dzielone wielokrotnie, aż staną się wystarczająco małe, aby podczas transformacji każdą z nich można było zaimplementować w jednym sprincie. Duża refaktoryzacja jest uznawana za ukończoną dopiero wtedy, gdy wykonane zostaną wszystkie mniejsze refaktoryzacje, na które została podzielona.

Tabela 1.1. Backlog usprawnień z refaktoryzacjami dla terminalu kontenerowego (fragment)

Nr	Refaktoryzacja	Wystarczająco mała na jeden sprint?
1.0	Wyodrębnienie kontekstu ograniczonego „planowanie sztauowania”	Nie
1.1	Wydzielenie części `PlanningService` z monolitu do wyspecjalizowanej wersji w kontekście ograniczonym „planowanie sztauowania”	Tak
1.2	Wydzielenie części `Container` z monolitu do wyspecjalizowanej wersji w kontekście ograniczonym „planowanie sztauowania”	Tak
2.0	Wyodrębnienie kontekstu ograniczonego „operacje składowania”	Nie
2.1	Wydzielenie części `PlanningService` z monolitu do wyspecjalizowanej wersji w kontekście ograniczonym „operacje składowania”	Tak
2.2	Przeniesienie kontekstu ograniczonego „operacje składowania” do jego własnej mikrousługi	Tak

W tabeli 1.1 refaktoryzacje są ponumerowane w sposób pokazujący, które z nich są ze sobą powiązane. W tym przykładzie refaktoryzacja 1.0 została podzielona na refaktoryzacje 1.1 i 1.2 (a także prawdopodobnie 1.x i 1.y, które pominięto dla zwięzłości).

Zespół decyduje, że wyodrębnione konteksty ograniczone będą nadal implementowane w Javie. Ponadto konteksty ograniczone zostaną najpierw zaimplementowane jako moduły w tej samej bazie kodu. Refaktoryzacja 2.2 pokazuje, że na późniejszym etapie rozważane jest przejście na architekturę mikrousługową.

W dodatkach znajduje się katalog różnych refaktoryzacji dziedzinowych, które pomagają w przeprowadzeniu transformacji.

Określanie priorytetów i realizowanie transformacji

Po wypełnieniu backlogu usprawnień zespół deweloperski planuje pierwszy sprint, w którym 30% wysiłku przeznacza na transformację. Pierwszą refaktoryzacją zaplanowaną na sprint jest 2.1 (Wydzielenie części `PlanningService` z monolitu do wyspecjalizowanej wersji w kontekście ograniczonym „operacje składowania”). Para programistów rozpoczynająca pracę nad 2.1 tworzy

pusty moduł `yardPlanning`, a w nim pustą klasę `yardPlanning.PlanningService`. Następnie zespół przenosi metodę `sendTransportOrder()` z monolitu do nowego modułu. Na razie nowy moduł pozostaje częścią tej samej bazy kodu co monolit. Później może zapaść decyzja o przeniesieniu nowego modułu do osobnego programu, a zatem przekształceniu go w mikrousługę.

Stary system jest przeprojektowywany w takich małych krokach. Odwiedzamy zespół wielokrotnie przez kolejne tygodnie i miesiące. Deweloperzy regularnie sprawdzają, co udało im się osiągnąć i jaki wpływ ich modyfikacje wywarły na strukturę systemu. Muszą nieustannie dostosowywać plan transformacji, dodawać nowe refaktoryzacje i modyfikować istniejące. Nie da się z góry przewidzieć wszystkich niezbędnych prac. W miarę wprowadzania zmian identyfikowane są kolejne zadania do wykonania.

Po kilku tygodniach widoczne są pierwsze pozytywne zmiany — w systemie pojawiają się obszary, w których łatwiej jest dodawać lub modyfikować funkcje. Po kilku miesiącach dalszy rozwój całego systemu staje się prostszy — zespołowi udało się rozwikłać wiele zależności i znacznie zredukować wysoki stopień sprzężenia w tej dawnej wielkiej bryle błota. Wreszcie, po latach, system staje się w pełni modularny — co za moment! Poszczególne konteksty ograniczone są wyraźnie rozpoznawalne w kodzie źródłowym. Wąskie interfejsy łączą moduły dziedzinowe, ale tylko tam, gdzie jest to konieczne.

Co się właśnie stało?

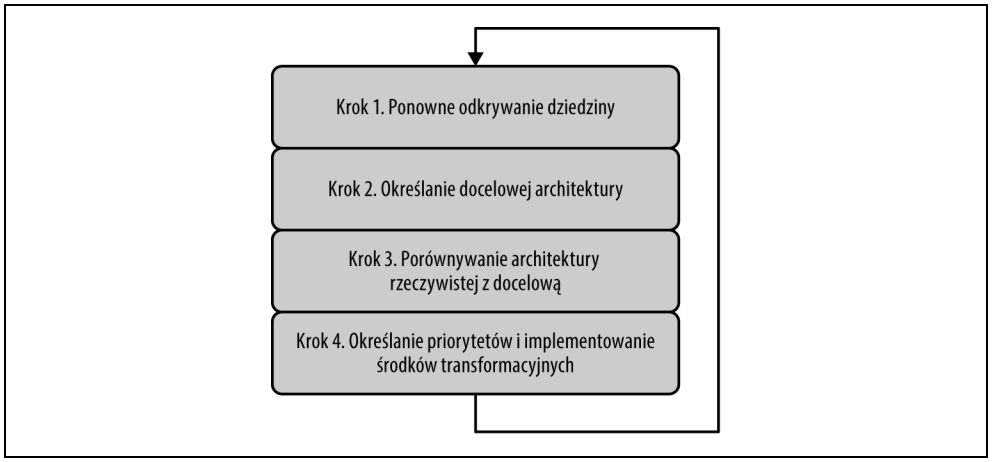
W tym wprowadzającym rozdziale przeprowadziliśmy Cię przez poszczególne etapy DDT w formie szybkiego przeglądu. Przedstawiliśmy wstępny zarys metody na przykładzie terminalu kontenerowego. Aby tak szybki przegląd był możliwy, znacznie uprościliśmy przykład.

Podczas transformacji rzeczywistego systemu oprogramowania pojawia się znacznie więcej trudności niż w tym uproszczonym przykładzie. Dziedzina jest znacznie większa i bardziej złożona, więc trzeba przeanalizować znacznie więcej procesów za pomocą historii dziedzinowych, a dekompozycja na konteksty ograniczone jest możliwa dopiero po wielu dyskusjach. W praktyce pierwsza dekompozycja jest często tylko punktem wyjścia do stopniowej transformacji, w której proces transformacji musi być przeprowadzany wielokrotnie — DDT jest procesem iteracyjnym zgodnym z filozofią Agile. Oprogramowanie w rzeczywistym świecie jest również znacznie większe i bardziej złożone niż w naszym małym przykładzie. Zazwyczaj składa się ze znacznie większej liczby części, a czasem z kilku systemów. Te części lub systemy są często bardzo mocno splątane, a rozdzielanie ich małymi krokami wcale nie jest oczywiste.

Aby dostarczyć więcej informacji o tym, jak można przeprowadzić DDT przy różnym stopniu chaosu w architekturze, w następnym podrozdziale odejdziemy od tego przykładu i opiszemy DDT jako metodę w połączeniu z różnymi typowymi architekturami, które spotykamy w praktyce.

Metoda

Metoda Domain-Driven Transformation składa się z czterech strategicznych kroków, które przedstawiono na rysunku 1.15.



Rysunek 1.15. Kroki metody Domain-Driven Transformation

Kroki te widziałeś już w przykładzie terminalu kontenerowego, ponieważ opis naszego podejścia do modernizacji TOS-u podzieliliśmy na odpowiednie podrozdziały:

Krok 1. Ponowne odkrywanie dziedziny (patrz rozdział 10.*)*

Odłóż istniejące rozwiązanie na bok i skup się na problemie z dzisiejszą wiedzą. Wykorzystaj metody modelowania kooperacyjnego, aby na nowo zrozumieć problem, jego dziedzinę i procesy. W przypadku TOS-u zastosowaliśmy metodę modelowania kooperacyjnego znaną jako Domain Storytelling — opowiadanie historii dziedzinowych (patrz podrozdział „Opowiadanie historii dziedzinowych" w rozdziale 4.).

Krok 2. Określanie architektury docelowej (patrz rozdział 11.*)*

Wyprowadź idealną architekturę docelową z nowo zdobytej wiedzy dziedzinowej. Wyznacz granice w kooperacyjnie wymodelowanych procesach i wyodrębnij docelową mapę kontekstową.

Krok 3. Porównywanie architektury rzeczywistej z docelową (patrz rozdział 12.*)*

Przeanalizuj rzeczywistą architekturę systemu i zidentyfikuj różnice względem architektury docelowej. Zbierz refaktoryzacje, które należy wykonać, aby przekształcić system ze stanu obecnego do docelowego.

Krok 4. Określanie priorytetów i implementowanie środków transformacyjnych (patrz rozdział 13.*)*

Zaplanuj i przeprowadź refaktoryzacje. Podziel duże refaktoryzacje na wystarczająco małe podrefaktoryzacje. Zidentyfikuj refaktoryzacje do następnej iteracji. Refaktoryzuj kod, a tym samym system, małymi krokami w kierunku lepszego stanu.

Te cztery kroki stanowią serce Domain-Driven Transformation, ale nasza metoda ma do zaoferowania znacznie więcej, a mianowicie wskazówki, jak postępować w zależności od stanu Twojej architektury.

Gdy klienci proszą nas o przeprowadzenie z nimi DDT, wielokrotnie stwierdzamy w kroku 3., podczas porównywania architektury rzeczywistej z docelową, że nie możemy od razu przejść do kroku 4. z transformacją kodu źródłowego w kierunku architektury docelowej wymodelowanej w kroku 2. Dzieje się tak, ponieważ systemy te są w bardzo złym stanie i wymagają pewnych prac przygotowawczych, zanim będzie można je zdekomponować. Czym są te prace przygotowawcze i jak je wybieramy?

Otóż kiedy porównujemy architekturę rzeczywistą z docelową w kroku 3., dla wszystkich systemów określamy MMI w skali od 0 do 10. We wstępnym przykładzie krótko przedstawiliśmy MMI jako wartość, która wskazuje jakość modularności systemu. MMI systemu TOS był dość niski i wynosił 4,9. W rzeczywistości MMI daje nam wskazówkę co do ścieżki, którą musimy obrać w przypadku prac przygotowawczych i transformacji.

Ścieżki przez DDT

MMI dzieli skalę od 0 do 10 na trzy obszary, w których znajdujemy typowe architektury:

MMI 0 – 4 (poziom czerwony) — wielka bryła błota

System bez struktury, w którym każdy komponent zna i wykorzystuje wszystkie inne.

MMI 4 – 8 (poziom żółty) — techniczny podział na warstwy

System z warstwami technicznymi, w którym istnieje kierunek użycia — od frontendu przez warstwę aplikacji i warstwę dziedziny do bazy danych.

MMI 8 – 10 (poziom zielony) — modularyzacja dziedzinowa

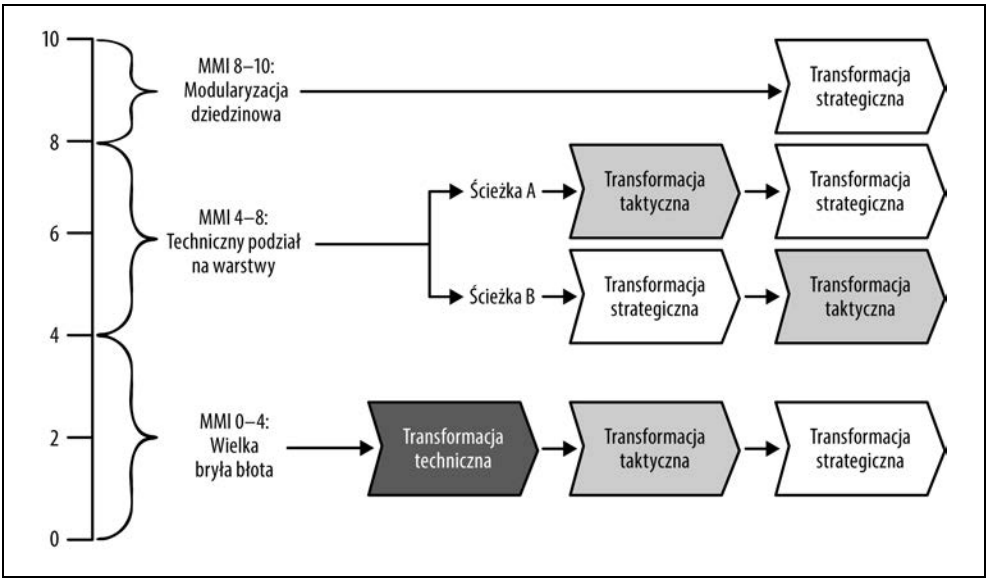
System podzielony na niezależne moduły zgodnie z dziedziną i jej poddziedzinami, z wąskimi interfejsami między modułami.

W kolejnych trzech podpunktach wyjaśnimy te trzy poziomy i przedstawimy nasze rekomendacje dotyczące odpowiednich dalszych działań w każdym przypadku. Przegląd wszystkich środków i ścieżek, które zostaną omówione poniżej, pokazano na rysunku 1.16.

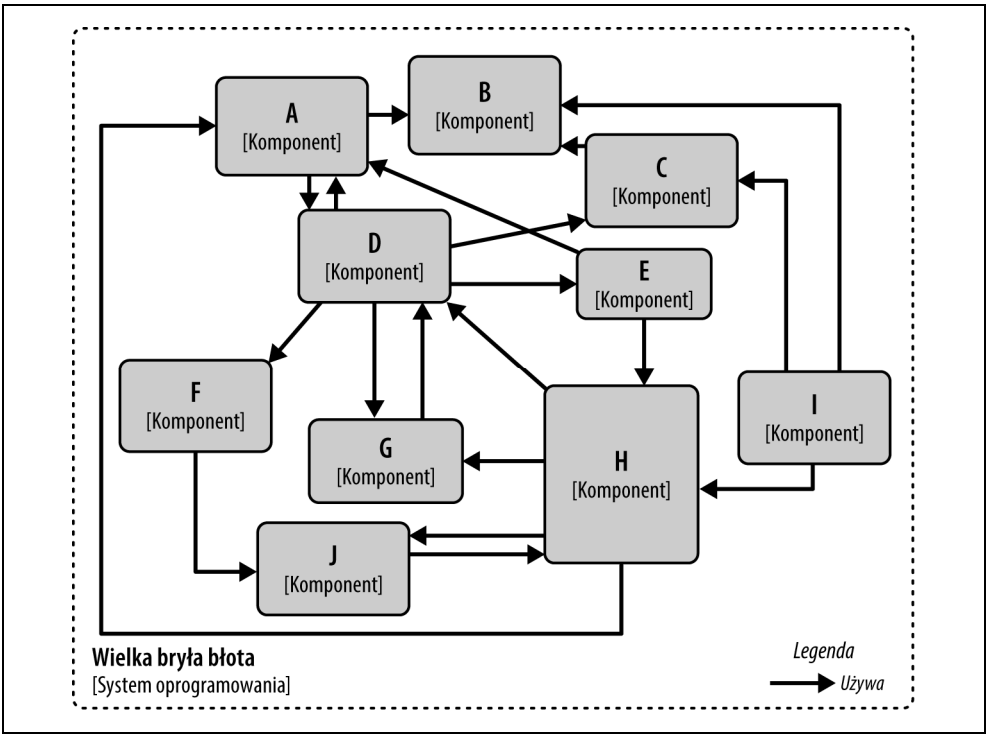
MMI 0 – 4 — wielka bryła błota

System z wartością MMI między 0 a 4 ma wszystkie negatywne cechy wielkiej bryły błota (patrz rozdział 5.). Reprezentację tej „architektury” pokazano na rysunku 1.17. W kodzie źródłowym i jego elementach nie można odnaleźć ani hierarchicznej struktury zgodnej z technicznym podziałem na warstwy, ani struktury odzwierciedlającej dziedzinę. Nie ma góry ani dołu, lewej ani prawej strony. Zamiast tego wszystko jest połączone ze wszystkim.

Ponieważ w systemach o MMI poniżej 4 trudno dostrzec jakiekolwiek rozpoznawalne struktury, a wszystko jest połączone ze wszystkim, ich projekt na poziomie klas jest słaby, a struktury na poziomie architektonicznym nie są modularne. Kod techniczny i logika biznesowa są



Rysunek 1.16. Ścieżki przez DDT



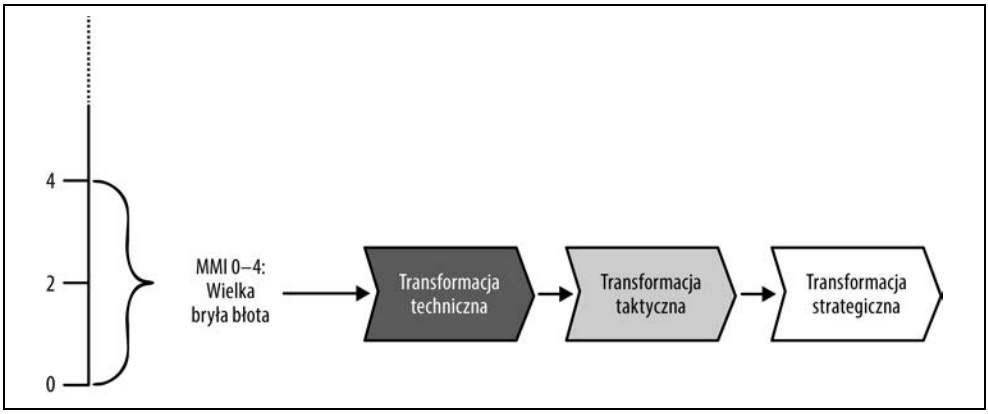
Rysunek 1.17. Wielka bryła błota — brak struktury poziomej i pionowej

ze sobą wymieszane, a identyfikacja klas, które mogłyby zostać przekształcone w model dziedzinowy, jest niemal niemożliwa. Ponadto takie systemy często mają wiele błędów oraz poważne problemy z bezpieczeństwem i infrastrukturą. Zanim w ogóle zaczniemy myśleć o podziale takiego systemu na moduły dziedzinowe przy użyciu czterech kroków Domain-Driven Transformation, czeka nas mnóstwo pracy.

Z naszego doświadczenia wynika, że system z MMI między 0 a 4 powinien najpierw zostać ustabilizowany technicznie (patrz rozdział 7.), a następnie wzmocniony wiedzą dziedzinową (patrz rozdział 8.). W naszej metodzie stabilizacja techniczna jest nazywana **transformacją techniczną**, a wzmacnianie wiedzą dziedzinową — **transformacją taktyczną**.

Transformacja techniczna obejmuje takie działania jak automatyzacja procesu budowania i wdrażania, aktualizacja przestarzałych platform i bibliotek, zwiększenie pokrycia testami w celu wykrywania błędów oraz poprawa odporności na błędy (patrz rozdział 7.). Transformacja taktyczna obejmuje działania takie jak wzbogacenie modelowania dziedzinowego, wprowadzenie obiektów wartości i projektowania kontraktowego (ang. *Design by Contract*) oraz redukcja cykli, zależności i dziedziczenia (patrz rozdział 8.). Szczegółowo omawiamy te działania w części II. Po zakończeniu transformacji technicznej i taktycznej można przejść do czterech kroków transformacji strategicznej, które opisujemy w części III.

Rysunek 1.18 przedstawia ścieżkę dla systemu z wartością MMI między 0 a 4 przez trzy obszary transformacji — techniczny, taktyczny i strategiczny.



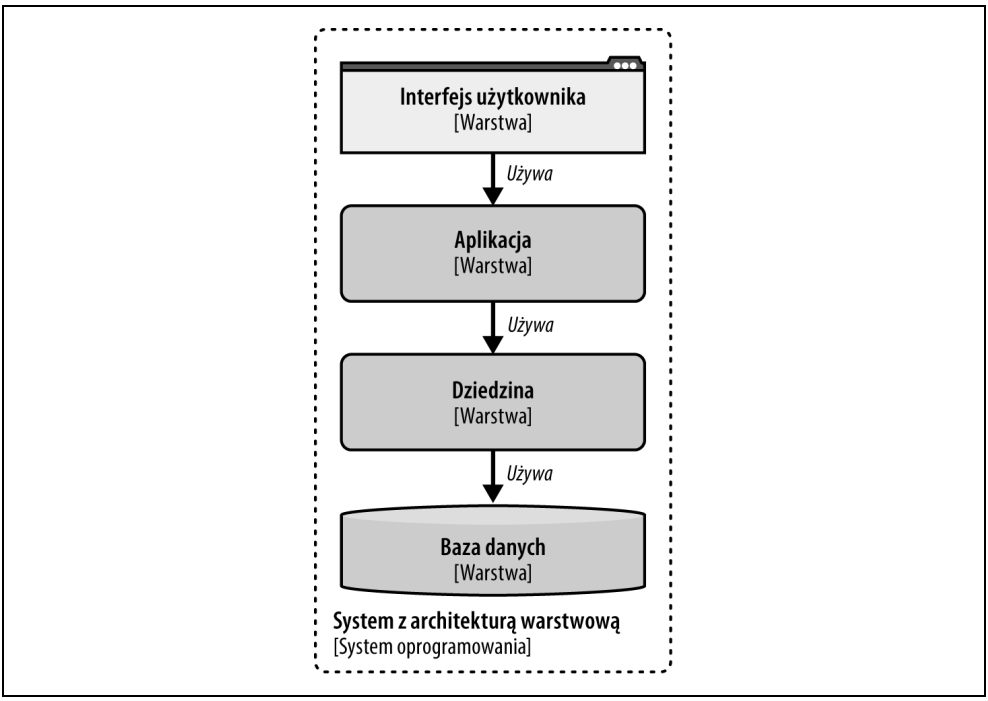
Rysunek 1.18. MMI i pierwsza ścieżka przez transformację

W kolejnych dwóch podpunktach dodajemy ścieżki, które należy obrać w przypadku systemów o wyższym wskaźniku MMI.

MMI 4 – 8 — architektura z warstwami technicznymi

W systemach z MMI między 4 a 8 podstawa techniczna jest zazwyczaj całkiem dobra i można w nich znaleźć typową architekturę warstwową, w której warstwy wywołują się wzajemnie od góry do dołu, czyli są acykliczne. Dalsze wyjaśnienia dotyczące architektury warstwowej i innych koncepcji architektonicznych można znaleźć w rozdziale 5.

Jak widać na rysunku 1.19, systemy z MMI między 4 a 8 mogą mieć warstwy zgodne z dekompozycją techniczną, ale pionowe przekroje według dziedziny (patrz rozdział 3.) są w dużej mierze nieobecne.



Rysunek 1.19. Architektura w warstwami technicznymi — istnieje struktura pozioma (tj. hierarchia techniczna), ale brak struktury pionowej (tj. dziedzinowej)

Jeśli napotkasz taki system, istnieją dwa sposoby wyjścia z tej sytuacji:

Ścieżka A

Najpierw transformacja taktyczna.

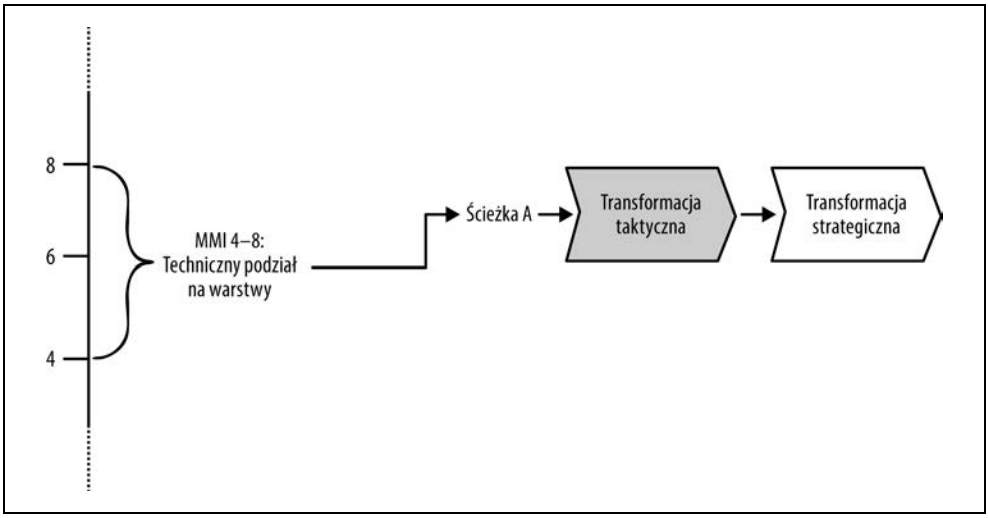
Ścieżka B

Najpierw transformacja strategiczna.

Wybór pierwszej lub drugiej ścieżki zależy od struktury warstwy dziedzinowej występującej w danym systemie.

Ścieżka A — najpierw transformacja taktyczna Powinieneś wybrać ścieżkę A, jeśli masz system z architekturą warstwową i wskaźnikiem MMI między 4 a 8, w którym warstwa dziedzinowa stanowi wielką bryłę błota. Taka sytuacja zwykle występuje wtedy, gdy struktura bazy danych została przeniesiona z tabel i ich relacji bezpośrednio do kodu źródłowego warstwy dziedziny. Rezultatem jest duża, splątana warstwa dziedzinowa bez jakiejkolwiek struktury odzwierciedlającej dziedzinę.

Idea tej ścieżki polega na tym, aby najpierw wzmocnić model dziedzinowy za pomocą taktycznej DDT (patrz rozdział 8.), a dopiero potem rozłożyć go na konteksty ograniczone z użyciem strategicznej DDT (patrz część III). Na rysunku 1.20 przedstawiono ścieżkę transformacji A dla wskaźnika MMI wynoszącego 4 – 8.



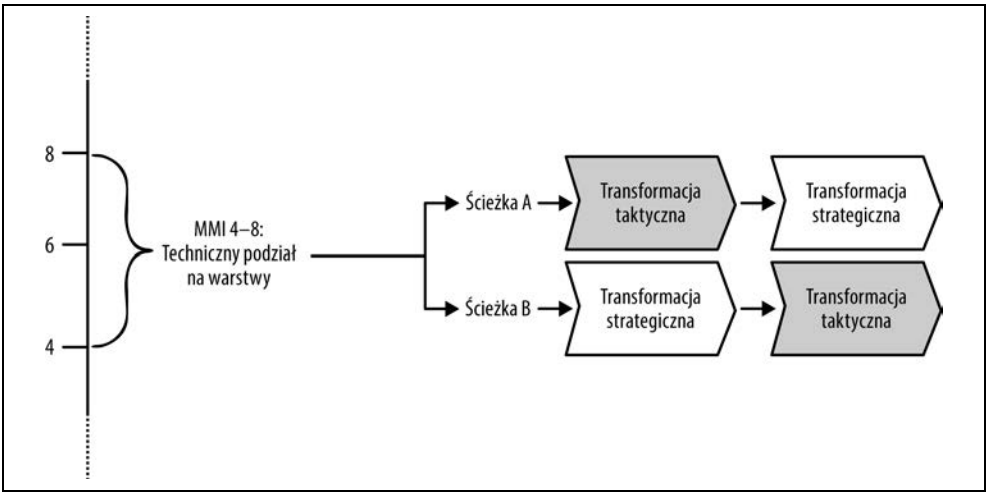
Rysunek 1.20. MMI i druga ścieżka transformacji

W przypadku dziedziny głównej (patrz rozdział 3.), czyli najważniejszego kontekstu ograniczonego w systemie, zalecamy zwrócenie szczególnej uwagi na wzmocnienie projektu wiedzą dziedzinową (patrz rozdział 8.). Dziedzina główna reprezentuje kluczowe wyróżniki biznesowe firmy na poziomie oprogramowania i musi być w dobrej kondycji, aby można ją było szybko i bezpiecznie rozbudowywać oraz dalej rozwijać. W przypadku poddziedzin pomocniczych i generycznych (ponownie patrz rozdział 3.), szczególnie tych, które rzadko się zmieniają, często można pominąć tę pracę.

Ścieżka B — najpierw transformacja strategiczna Jeśli masz system z MMI na poziomie 4 – 8, w którym warstwa dziedzinowa nie jest wielką bryłą błota, lecz poza kilkoma mniejszymi cyklami zależności ma rozpoznawalną strukturę, możesz zacząć od strategicznej DDT pomimo złego projektu, a dopiero potem go ulepszać.

W podejściu tym najpierw dekomponuje się system na konteksty ograniczone z użyciem transformacji strategicznej (patrz część III), nawet jeśli struktura logiki biznesowej jest niezadowalająca.

Po ustaleniu kontekstów ograniczonych w architekturze można przejść do drugiego kroku i wzmocnić słabo ustrukturyzowaną logikę biznesową wiedzą dziedzinową poprzez transformację taktyczną (patrz rozdział 8.). Na rysunku 1.21 dodajemy ścieżkę B dla systemów z MMI na poziomie 4 – 8.



Rysunek 1.21. MMI i trzecia ścieżka transformacji

Ścieżka A czy ścieżka B? Przejścia od dużej, splątanej warstwy dziedzinowej do modelu, w którym można rozpoznać początki struktury dziedzinowej, są zazwyczaj płynne, więc w niektórych przypadkach nie da się jednoznacznie określić, czy należy podążać ścieżką A, czy można wybrać ścieżkę B.

W takiej sytuacji zazwyczaj zalecamy ścieżkę B. Zaczynamy więc od czterech kroków strategicznej DDT (patrz część III) i dopiero po podzieleniu i dekompozycji systemu na konteksty ograniczone zaczynamy wzmacniać kod wiedzą dziedzinową. W przeciwnym razie powstające klasy dziedzinowe są bardzo duże (czyli monolityczne), mimo że odzwierciedlają dziedzinę. Jak już wspomnieliśmy, wybór nie musi być czarno-biały — na przykład po wyodrębnieniu pierwszego kontekstu ograniczonego może mieć sens praca nad tym konkretnym kontekstem z wykorzystaniem transformacji taktycznej, zanim wyodrębnimy kolejny kontekst ograniczony.

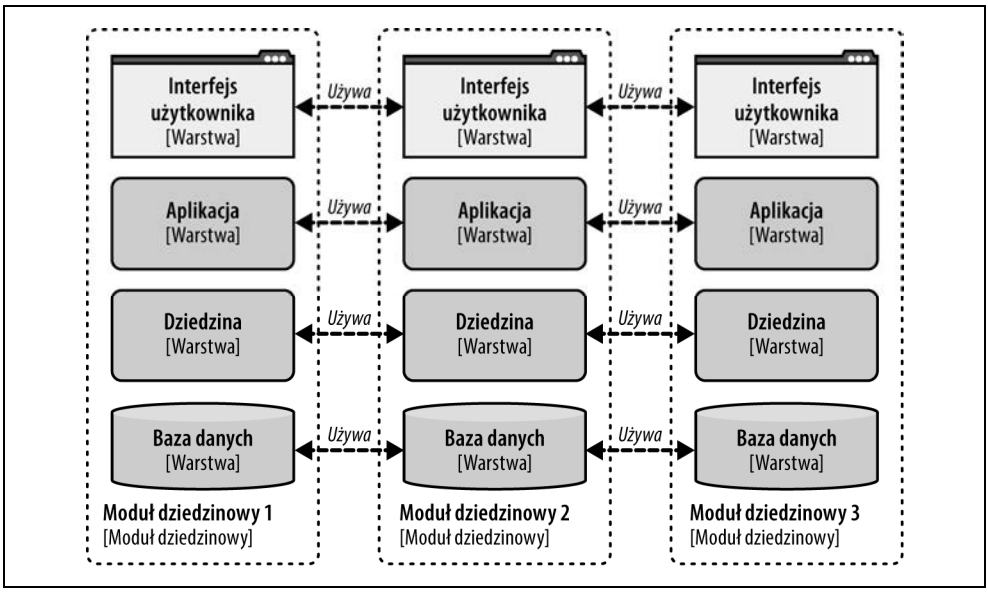
Podobna kombinacja obu ścieżek może być przydatna, gdy zaczynasz od ścieżki A: najpierw przeprowadzasz kilka przygotowawczych kroków taktycznych (np. redukcja sprzężenia poprzez podział rekordu aktywnego na agregat i repozytorium), następnie wydzielasz część za pomocą kroków strategicznych (np. wyodrębniasz kontekst ograniczony), a potem ponownie stosujesz kroki taktyczne (np. walczysz z anemią modelu w nowo wyodrębnionym kontekście ograniczonym).

W kolejnych rozdziałach poznasz studia przypadków, które *zilustrują obie ścieżki*:

- W rozdziale 4. przedstawimy studium przypadku z dziedziny *bankowej*, do którego wrócimy w rozdziale 8., ponieważ dla tego przypadku wybraliśmy *ścieżkę A — najpierw transformacja taktyczna*. W studium tym najpierw transformujemy taktycznie, a potem strategicznie.
- W części III pojawi się studium przypadku z dziedziny *leasingu samochodowego*. W tym studium wybraliśmy *ścieżkę B — najpierw transformacja strategiczna*. Najpierw transformujemy strategicznie, a potem ulepszamy taktycznie.

MMI 8 – 10 — modularyzacja dziedzinowa

Jeśli system uzyskał ocenę między 8 a 10 w skali MMI, oznacza to, że ma już modularną strukturę, która odzwierciedla dziedzinę. Architektura systemu będącego w tak dobrym stanie powinna schematycznie wyglądać tak, jak pokazano na rysunku 1.22.



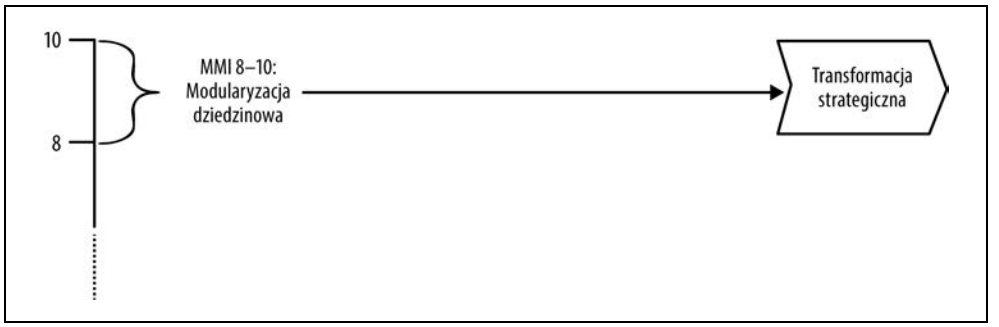
Rysunek 1.22. Modularyzacja dziedzinowa — warstwy techniczne wewnątrz modułów dziedzinowych

Dwukierunkowe strzałki między modułami na rysunku 1.22 wskazują, że nawet w przypadku tych dobrze uformowanych systemów istnieją pewne relacje, które wymagają uporządkowania. Może być również konieczne przeniesienie niektórych klas, aby dekompozycja dziedzinowa była naprawdę udana. Często moduły dziedzinowe zaimplementowane w systemie nie są jeszcze prawdziwymi kontekstami ograniczonymi.

Dla takich systemów warto przeprowadzić cztery kroki strategicznej DDT, ponieważ ich struktura jest już bliska docelowej, a trwałą dekompozycję można osiągnąć przez wprowadzenie zaledwie kilku usprawnień.

Na rysunku 1.23 przedstawiono ścieżkę dla systemów o MMI w zakresie 8 – 10. Kompletny diagram dla wszystkich trzech poziomów MMI: 0 – 4 (wielka bryła błota), 4 – 8 (techniczny podział na warstwy) i 8 – 10 (modularyzacja dziedzinowa) znajduje się na początku tego podrozdziału.

Szczegółowy opis czterech kroków transformacji strategicznej wraz z licznymi przykładami i dokładnymi wyjaśnieniami znajduje się w części III.



Rysunek 1.23. MMI i czwarta ścieżka transformacji

Podsumowanie

W tym rozdziale przedstawiliśmy przegląd naszej metody Domain-Driven Transformation i opisaliśmy różne procedury w zależności od stanu systemu i jego aktualnej architektury. Po tym podstawowym, nieco abstrakcyjnym wprowadzeniu możesz przejść do odpowiednich sekcji:

- Jeśli brakuje Ci podstaw lub części podstaw, zajrzyj do poszczególnych rozdziałów z części I.
- Jeśli podejrzewasz, że Twój system oprogramowania ma MMI poniżej 4, przeczytaj środki zaradcze opisane w części II i zastosuj je w swoim systemie.
- Jeśli chcesz zająć się transformacją strategiczną, zacznij od części III.
- Aby transformacja strategiczna zakończyła się sukcesem, zwykle trzeba zmienić nie tylko strukturę systemu, ale także strukturę zespołów, które ten system rozwijają. *Transformacja organizacyjna zespołów* jest tematem rozdziału 9.

Miłej lektury!

Skorowidz

A

adnotacje, 218
agregat, 213, 214, 260, 378
analityk architektury, 241, 264, 275
analiza wymagań, 50
  metoda
    Domain Storytelling, 23, 98–106
    EventStorming, 44, 106–116
    Scenario Casting, 116–131
anemiczny model dziedzinowy, 202
architektura oprogramowania, 135
  cebulowa, 154
  czterowarstwowa DDD, 82
  czysta, 154
  dziedzinowa, 50
  heksagonalna, 153
  jawna, 154
  mikrousługowa, 149
  monolityczna, 149
  rzeczywista, 283, 284
  rzeczywista a sztuczna inteligencja, 282
  rzeczywista i docelowa, 285
  Self-contained Systems, SCS, 162
  sześciokątna, 156
    repozytorium i adaptery, 157
    warstwa aplikacji, 158
    warstwa dziedziny, 158
    warstwa infrastruktury, 158
    warstwa interfejsu użytkownika, 158
  techniczna, 50
  trójwarstwowa, 153
  warstwowa, 148, 152
  według przypadków użycia, 287, 288
  z warstwami technicznymi, 37
  zorientowana na usługi, SOA, 148, 158–164
asystent kodowania
  GitHub Copilot, 220
  JetBrains AI Assistant, 220

B

backend, 223
backlog
  produktu, 309
  usprawnień, 309
bank, *Patrz także* studium przypadku
  anemiczny model dziedzinowy, 202
  dziedziczenie, 216
  eksploracja chaotyczna, 108
  granice, 111
  jMolecules, 218
  mikrousługi SOA, 160, 161
  model dziedzinowy, 115
  oś czasu, 109
  projektowanie kontraktowe, 209
  refaktoryzacja nieograniczonego modelu dziedzinowego, 289
  referencje tożsamościowe, 213
  Software Design EventStorming, 114
  tożsamość, 211
  wprowadzanie obiektów wartości, 205
  zespół ds. skomplikowanego podsystemu, 230
  zróżnicowane elementy dziedzinowe, 346
baza danych, 223
BBoM, Big Ball of Mud, 143
błędy, 193, 195
budowniczy, Builder, 86
burza zdarzeń, EventStorming, 106–116

C

C4
  diagram komponentów modelu, 223
  diagram kontekstowy systemu, 56, 58
  kod, 139
  komponenty, 139
  kontekst, 137
  kontenery, 138
casting scenariuszowy, Scenario Casting, 116–131
cykle i zależności, 212

D

Data-Driven Design, DDD, 43
  architektura warstwowa, 82
  projektowanie taktyczne, 82
  spójność i sprzężenie, 142
dekompozycja
  dziedziny, 261
    wsparcie AI, 262
  interfejsu użytkownika, 297, 298
dokumentowanie architektury, 136, 218
Domain Storytelling, 23, 98–106
  czynniki zakresu, 103
  czysta, 104
  język dziedzinowy, 105
  notacja, 98
  rysowanie granic, 101
  scyfryzowana, 103
Domain-Driven Design, DDD, 31, 52, 54
  a dziedzina biznesowa, 56
  a rozwiązanie programowe, 70
  architektury wewnętrzno-zewnętrzne, 154
  elementy konstrukcyjne, 156
  eliminowanie przypadkowej złożoności, 55
  język wszechobecny, 58
  kontekst bańkowy, 76
  mapowanie kontekstów, 72
  model dziedzinowy, 77
  poddziedziny, 66
  projektowanie strategiczne, 64, 70
  zdarzenia dziedzinowe, 88
Domain-Driven Transformation, DDT, 34, 239, 240, 352, 354
  dopasowanie architektury rzeczywistej do docelowej, 25, 34, 278
    identyfikowanie rzeczywistej architektury, 281
    lista refaktoryzacji, 302
    mapa kontekstowa a kod źródłowy, 284
  język wszechobecny, 58, 251, 372
  kontekst ograniczony, 70, 79, 269
  modelowanie architektury docelowej, 24, 34, 264, 274
    klasyfikowanie poddziedzin, 270
    mapa kontekstowa, 266
    określanie relacji, 274
    przypisywanie kontekstów do zespołów, 273
  określanie MMI, 35
  określanie priorytetów i realizowanie transformacji, 32, 34, 305
    dodawanie refaktoryzacji taktycznych, 309
    planowanie wykonalnych refaktoryzacji, 310
    wybór refaktoryzacji strategicznej, 307
    wyodrębnianie kontekstu, 313
  ponowne odkrywanie dziedziny, 21, 34, 243
    modelowanie stanu obecnego, 248
    optymalizacja procesów biznesowych, 252
    wyodrębnianie wiedzy dziedzinowej, 248
    znajdowanie poddziedzin, 252
  ścieżki, 36–42
  transformacja
    taktyczna, 199–221
    techniczna, 181–198
    zespołowo-organizacyjna, 222–241
  użycie metodyki Kanban, 311
dopasowanie architektury rzeczywistej do docelowej, 25, 34, 278
dziedziczenie, 216
dziedzina
  biznesowa, 56
  główna, 67
    wykres, 67
  implementacja modelu, 345
  kształt, 349
  obiektów materialnych, 350
  przepływ pracy, 342
  w pełni cyfrowa, 350
dziedzinowa transformacja organizacyjna, 226

E

ekspert dziedzinowy, 244, 246
eksperymenty myślowe, 249
elementy
  dziedzinowe
    centralny, 346
    zróżnicowane, 346
  konstrukcyjne, 83, 154, 156
eliminowanie
  cykli i zależności, 212
  przypadkowej złożoności, 55
  zależności, 190
encje, 83, 211
  duplikowanie, 318
  leczenie anemii, 373
  określanie tożsamości, 211
  przenoszenie tożsamości, 322
  tożsamość, 321
  wyodrębnianie, 380
EventStorming, 44, 106–116
  eksploracja chaotyczna, 107
  format Big Picture, 107
  kodowanie kolorami, 112
  porządkowanie zdarzeń, 109
  rysowanie granic, 111
  Software Design EventStorming, 113
  znajdowanie modelu dziedzinowego, 115

F

fabryka abstrakcyjna, Abstract Factory, 86
frontend, 223

G

graf obiektów, 87

H

hierarchiczność, 166
historie dziedzinowe, Domain Storytelling, 98–106

I

implementacja repozytorium, 379
indeks dojrzałości modularności, MMI, 27, 164
interfejs
  repozytorium, 379
  użytkownika
    dekompozycja, 298
    monolityczny, 299
    w kontekście, 299, 300
    w kontekście z integracją, 301
    z integracją i zdarzeniami, 301

J

język
  dziedzinowy, 105
  wszechobecny, 58, 251, 372
    słowniczki, 61
    terminy dziedzinowe, 59
    wszystko po angielsku, 63
    zbędne terminy technologiczne, 61
jMolecules, 218

K

kino
  agregaty, 85
  czysta historia dziedzinowa, 104
  domena główna, 68
  duplikacja i redundancja, 81
  encje, 83
  graf obiektów, 87
  granice w historiach dziedzinowych, 101
  historie dziedzinowe, 98
  język dziedzinowy, 105
  język wszechobecny, 58
  komponenty, 139
  kontekst, 137
  konteksty ograniczone, 71, 79
  kontenery, 138
  mapa kontekstowa, 71
  mapowanie kontekstów, 75
  model dziedzinowy, 77, 79
  nowa domena główna, 69
  obiekty wartości, 84
  poddziedziny, 64–66
  podobieństwo strukturalne, 78
  projektowanie kontraktowe, 196
  repozytorium, 86
  scyfryzowana historia dziedzinowa, 103
  słowniczek, 61
  synonimy, 60
  system Cine2001, 55, 56

kino
  system Cine2001 z otoczeniem, 58
  terminologia dziedzinowa, 59
  topologie zespołów, 232
  tryby interakcji zespołów, 232
  usługa aplikacyjna, 86
  usługi dziedzinowe, 85
  wskaźniki, 256
  wykres dziedziny głównej, 69
  zbędne terminy technologiczne, 62
  zdarzenia dziedzinowe, 89, 90
  zespoły strumieniowe, 228
  zespół platformowy, 229
  zespół wspomagający, 230
  zła dekompozycja, 258
klasyfikowanie poddziedzin, 270
kod
  biznesowy, 200
  techniczny, 200
koncepcje architektoniczne, 53
konsolidator, 191
kontekst
  bańkowy, 76
  ograniczony, 70, 79, 269
    implementacja od zera, 358
    przypisywanie do zespołów, 273
    typy współpracy, 72
    wyodrębnianie, 357
  wzorzec mapowania
    język opublikowany, 74
    klient-dostawca, 73
    konformista, 73
    oddzielne drogi, 74
    partnerstwo, 73
    usługa otwartego hosta, 73
    warstwa przeciwskażeniowa, 73
    wielka bryła błota, 74
    współdzielone jądro, 74
kontrakt, 377

L

leasing samochodów, 245
  analiza rzeczywistej architektury, 282, 285
  dekompozycja interfejsu użytkownika, 297
  duplikowanie encji, 318
  historia dziedzinowa, 250
  klasyfikacja, 271
  lista refaktoryzacji, 303
  lista ustaleń, 303
  mapa kontekstowa, 266, 277
  określanie priorytetów refaktoryzacji, 309
  optymalizacja procesu, 252
  planowanie refaktoryzacji w backlogu sprintu, 311
  ponowne odkrywanie dziedziny, 247
  praca z ustaleniami, 337
  przenoszenie tożsamości encji, 322
  przydział zespołów, 273
  refaktoryzacja anemicznego modelu dziedzinowego, 330
  refaktoryzacja nieograniczonego anemicznego modelu dziedzinowego, 291
  relacje, 275
  role i uprawnienia, 268
  rozplątywanie modelu danych, 333
  wprowadzanie zdarzenia, 323
  wskaźniki, 254
  wyodrębnianie kontekstu, 313
luźne sprzężenie, 142

M

mapa kontekstowa, 71, 266, 284, 285
mapowanie kontekstów, 266
  wzorce, 73
metoda fabryczna, Factory Method, 86
metody ustawiające
  usuwanie, 375
  zastępowanie, 374
metodyka Kanban, 311
metryki sprzężenia, 192
miary rozmiaru, 192
mikrofrontendy, 299
mikrousługi, 149, 150
  SOA, 161
MMI, Modularity Maturity Index, 27, 164
  architektura z warstwami technicznymi, 37
  hierarchiczność, 166
  kryteria, 167, 168
  modularność, 165
  modularyzacja dziedzinowa, 35, 41
  obliczanie wskaźnika, 167, 169
  spójność wzorców, 166
  techniczny podział na warstwy, 35
  wielka bryła błota, 35

model
anemiczny dziedziny, 329
C4, 21, 136
danych, 333
danych nieograniczony, 297
dziedzinowy, 77, 79
duplikacja, 80
formularzowy, 348
konteksty ograniczone, 78, 80
nieograniczony, 286, 288
nieograniczony anemiczny, 291
podobieństwo strukturalne, 78
redundancja, 80
wielokrotny użytek, 80
kanoniczny, 162
modelarz dziedziny, 241, 244, 264, 266, 271
modele nieograniczone, 286–298
modelowanie
architektury docelowej, 24, 34, 264
dziedziny, 50
kooperacyjne, Collaborative Modeling, CoMo, 44, 52, 91
jako filar DDD, 97
koncepcje, 95
łączenie osób, 94
metoda Domain Storytelling, 23, 93, 98–106
metoda EventStorming, 44, 106–116
metoda Example Mapping, 93
metoda Scenario Casting, 93, 116–131
metoda Storystorming, 93
metoda User Story Mapping, 93, 133
monopolizacja modelu, 92
wsparcie AI, 133
wybór metody, 131
zdalne, 133
w kodzie, 54
zespołowe, 23
model-widok-kontroler, Model-View-Controller, MVC, 158
model-widok-model widoku, Model-View--ViewModel, MVVM, 158
model-widok-prezenter, Model-View-Presenter, MVP, 158
modernizowanie architektury, 234
modularność, 165
modularyzacja dziedzinowa, 35, 41
moduły, 140
spójność i sprzężenie, 140
monolit, 148
monopolizacja modelu, 92

N

narzędzie do przeglądu architektury, 27

O

obiekty wartości, 205, 207
obszary decyzyjne, 49
ograniczanie
dziedziczenia, 215
sprzężenia, 212
złożoności, 55
złożoności przypadkowej, 50, 55, 148
określanie
docelowej architektury, 24, 34, 264
priorytetów i realizowanie transformacji, 32, 305
opiekun produktu, product owner, PO, 57
opowiadanie historii dziedzinowych, Domain Storytelling, 23, 98–106
optymalizacja procesów biznesowych, 252
organizacja
horyzontalna, 223
wertykalna, 225

P

piramida testów, 188
poddziedziny, 64
generyczne, 66, 67
pomocnicze, 66, 67
podział
rekordu aktywnego, 378
repozytorium, 379
polimorfizm, 191
pomocnik transformacyjny
Domain-Driven Design, 31
indeks dojrzałości modularności, MMI, 28, 164
model C4, 21
narzędzia do przeglądu architektury, 27
opowiadanie historii dziedzinowych, 23, 98–106
ponowne odkrywanie dziedziny, 21, 34, 243

porównanie architektury rzeczywistej
z docelową, 25, 34, 278
projekt strategiczny
kontekst ograniczony, 70
mapa kontekstowa, 71
projektowanie
dziedzinowe, DDD, 31, 43, 54
kontraktowe, Design by Contract, 196, 208
strategiczne, 64, 70, 71
taktyczne, 82
agregat, 85
elementy konstrukcyjne, 82
encja, 83
fabryka, 86
obiekt wartości, 84
repozytorium, 85
usługa aplikacyjna, 86
usługa dziedzinowa, 84
wzorce projektowe, 83
przenoszenie
logiki do encji, 376
logiki dziedzinowej, 329
lokalnych danych do kontekstu, 318
metod pobierających do kontekstu, 318
metod ustawiających do kontekstu, 318
tożsamości encji, 322
przestrzeń
problemu, 46
rozwiązań, 46
pulpity nawigacyjne, 192
punkt dostępowy, 191

R

rdzeń aplikacji, 200
refaktoryzacja, 189
dziedzinowa, 355
eliminowanie zależności, 190
modelu dziedzinowego
anemicznego, 330
anemicznego nieograniczonego, 291
nieograniczonego, 289
określanie priorytetów, 309
planowanie, 311
przerywanie, 310
socjotechniczna, 366
strategiczna, 356, 360
implementowanie kontekstu
ograniczonego od zera, 358
wyodrębnianie kontekstu ograniczonego,
357
szacowanie, 311
taktyczna, 220, 309, 360, 371
użycie miar, 192
referencje tożsamościowe, 213
relacje, 275
repozytorium, 378, 379
rozplątywanie modelu danych, 333
rozwiązanie programowe, 70

S

Scenario Casting, 116–131
druga runda, 130
kolejne sesje, 127
przebieg warsztatu, 117
warsztaty
burza mózgów, 119
składanie, 125
skupienie, 122
SCS, Self-contained Systems, 162
SOA, Service-Oriented Architecture, 148, 158–164
Software Design EventStorming, 113, 114
spoina, 190
konsolidacyjna, 191
obiektowa, 191
preprocesowa, 191
spójność, 140
wzorców, 166
sprzężenie, 141, 212
strategiczna transformacja dziedzinowa,
Patrz Domain-Driven Transformation
studium przypadku
bank, 108
kino, 55
leasing samochodów, 245
Scenario Casting, 118
terminal kontenerowy, 46
system
operacyjny terminalu, TOS, 20
samodzielny, SCS, 162
socjotechniczny, 222

systemy zastane
  aktualizacja
    bibliotek i platform, 183
    IDE, 181
    języka programowania, 184
  automatyzacja budowania i wdrażania, 182
  eliminowanie ostrzeżeń kompilatora, 185
  porządkowanie kodu na bieżąco, 188
  problemy, 179
  przekształcanie, 176
  refaktoryzacja, 189
  style architektoniczne, 148
  wymiana
    krok po kroku, 174
    metodą wielkiego wybuchu, 171
  zwiększanie
    odporności na błędy, 193
    pokrycia testami, 185

T

techniczny podział na warstwy, 35, 37
terminal kontenerowy
  architektura docelowa, 29, 31
  centralny element dziedzinowy, 346
  narzędzie do przeglądu architektury, 30
  określanie docelowej architektury, 24
  określanie priorytetów i realizowanie transformacji, 32
  ponowne odkrywanie dziedziny, 21
  porównanie architektury rzeczywistej z docelową, 25
  proces terminalu kontenerowego, 26
  refaktoryzacje, 32
  TOS, Terminal Operating System, 20, 28
  TOS w kontekście, 22
  złożoność w przestrzeni problemu i rozwiązania, 46
  złożoność zasadnicza i przypadkowa, 47
testy, 185
  integracyjne, 187
  jednostkowe, 187
  UI, 187
topologie zespołów, 227
transformacja
  starego systemu, 170–177
  strategiczna dziedzinowa, *Patrz* Domain-Driven Transformation
  taktyczna, 37, 38, 199, 354
  techniczna, 37, 181, 354
  zespołowo-organizacyjna, 222, 354, *Patrz także* zespoły

U

usługi
  silnie sprzężone, 327
  wyodrębnianie, 381
usprawnianie organizacji pracy, 222

W

wartość
  całościowa, whole value, 84
  null, 193
wiedza dziedzinowa, 199, 202
wielka bryła błota, BBoM, 35, 36, 74, 143
  degradacja systemu, 146
  na poziomie architektury, 144
  na poziomie klasy, 143
wskaźniki granic poddziedzin, 253
wyjątki, 194, 197
wyodrębnianie
  encji, 380
  kontekstu ograniczonego, 357
  usługi, 381
  wyspecjalizowanej
    encji, 362
    tabeli, 364
    usługi, 361
wzorce
  mapowania kontekstu, 73
  projektowe, 83, 86, 158
wzorzec dziedzinowy
  Dialog, 344
  Potok, 342
  Tablica, 343

Z

zasada podstawienia Liskov, 215
zastąpienie typu prostego obiektem, 372
zdarzenia, 323-324
  dziedzinowe, 88, 327

zespoły
AMET, 241
backendowe, 223
bazodanowe, 223
ds. skomplikowanego podsystemu, 230
ewolucja w czasie, 233
frontendowe, 223
międzyfunkcyjne, 226, 367, 368
platformowe, 224, 229
reorganizacja, 226
strumieniowe, 228
tryby interakcji, 231
typy topologii, 228
warstwowe, 367–369
wspomagające modernizację architektury, 230, 234
zorganizowane horyzontalnie, 225
zintegrowane środowisko programistyczne, IDE, 181
zła dekompozycja, 258
złożoność, 45
modelowanie kooperacyjne, 91
ograniczanie
Domain-Driven Design, 52, 54
koncepcje architektoniczne, 53, 135
modelowanie kooperacyjne, 52, 91
przestrzeń problemu, 46
przestrzeń rozwiązań, 46
przypadkowa, 45, 47
eliminowanie, 50, 55, 148
źródła, 48
w systemach zastanych, 51
zasadnicza, 45, 47

Ź

źródła
złożoności, 46
złożoności przypadkowej, 48

Przekształć legacy w przewagę konkurencyjną dzięki Domain-Driven Design

Modernizacja systemów zastanych to jedno z największych wyzwań współczesnej inżynierii oprogramowania. W erze cyfrowej transformacji firmy borykają się z przestarzałymi monolitami, które hamują innowacje i generują coraz wyższe koszty utrzymania. Domain-Driven Design (DDD), podejście od lat uznawane za skuteczne w tworzeniu nowego oprogramowania, przechodzi drugą młodość jako metodyka transformacji systemów legacy. Książka wypełnia istotną lukę na rynku — oferuje kompleksowe, praktyczne podejście do modernizacji opartej na zasadach DDD, łączące aspekty techniczne, biznesowe i organizacyjne.

Autorzy prezentują metodę Domain-Driven Transformation (DDT) na poziomie strategicznym i organizacyjnym. Prowadzą czytelnika przez cztery kluczowe kroki: ponowne odkrywanie dziedziny biznesowej, modelowanie architektury docelowej, dopasowanie architektury rzeczywistej do docelowej, a także określanie priorytetów i realizację działań transformacyjnych. Szczególną wartość stanowi indeks dojrzałości modularności (MMI), który pozwala ocenić stan systemu i wybrać odpowiednią ścieżkę transformacji. Książka pokazuje, jak unikać pułapek wielkiego przepisania, stosować refaktoryzacje dziedzinowe i reorganizować zespoły według metodologii team topologies.

Jeśli rozpoczynasz projekt transformacji, polecam przeczytać tę książkę!

Michael Feathers, autor książki
Praca z zastanym kodem

To najlepsza książka o DDD dostępna na rynku

Sergio Morazán, OBI Group Holding

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-3881-6
9 788328 938816
Cena: 129,00 zł

Najważniejsze zagadnienia:

- Ocena kondycji systemu i wybór właściwej ścieżki transformacji
- Techniki ponownego odkrywania dziedziny
- Katalog refaktoryzacji dziedzinowych
- Przekształcenie zespołów zorganizowanych horyzontalnie w zespoły zorganizowane wertykalnie
- Praktyczne studia przypadków z branży kinowej, bankowej, leasingowej i portowej
- Zastosowanie AI do wsparcia procesu transformacji i refaktoryzacji

Carola Lilienthal — dyrektor generalna WPS – Workplace Solutions w Hamburgu, architektka oprogramowania i autorka książki *Sustainable Software Architecture*. Przeanalizowała ponad 300 systemów dla klientów z całego świata.
Henning Schwentner — programista, konsultant i instruktor systemów w WPS. Współautor książki *Domain Storytelling*, jest liderem opinii w społeczności DDD. Oboje regularnie występują na międzynarodowych konferencjach.

Wypełnia lukę w literaturze poświęconej projektowaniu dziedzinowemu, koncentrując się na praktycznym zastosowaniu zasad DDD w transformacji systemów zastanych

Gien Verschatse, współautor *Collaborative Software Design*