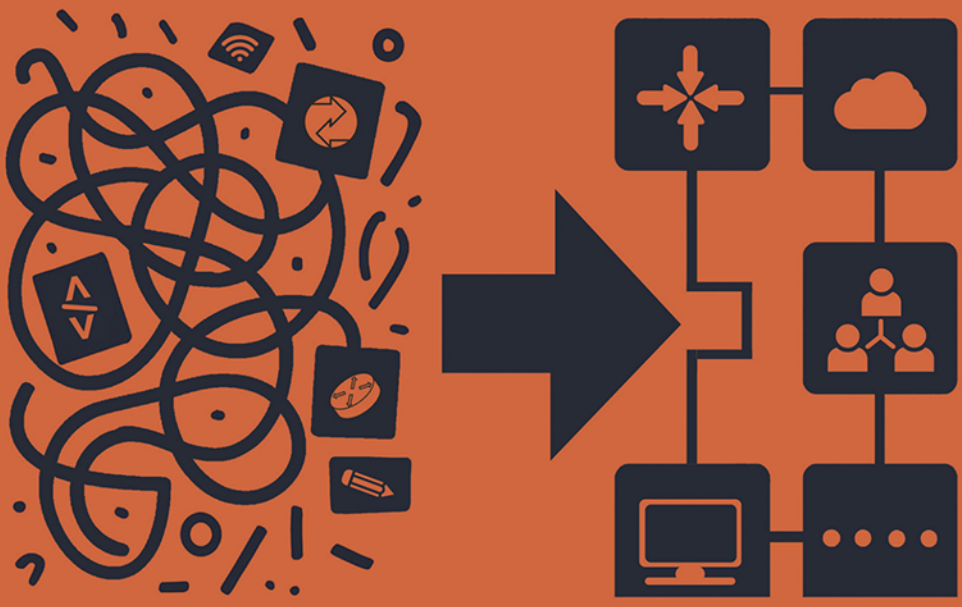
packt

Refaktoryzacja domenowa

Przewodnik DDD po przekształcaniu architektury monolitycznej w systemy modularne i mikrousługi



Alessandro Colla, Alberto Acerbis

Helion

Tytuł oryginału: Domain-Driven Refactoring: A hands-on DDD guide to transforming monoliths into modular systems and microservices

Tłumaczenie: Anna Mizerska

ISBN: 978-83-289-3327-9

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
helion.pl/user/opinie/refdom
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

CZĘŚĆ 2. Refaktoryzacja odziedziczonych systemów

Rozdział 3

Wzorce strategiczne

Jak dowiedziałeś się w poprzednich rozdziałach, największym wyzwaniem na początku każdego projektu — czy to nowego, czy refaktoryzacji — jest język. Do tej pory powinna być już oczywista waga zrozumienia domeny i zdefiniowania wspólnego języka do budowania wiedzy. Jednak tworzenie języka wszechobecnego wiąże się z ukrytymi wyzwaniami, często wynikającymi z ludzkich sposobów zachowania.

Teraz, gdy wiesz już, jak unikać tych błędów, jesteś gotowy odkryć wzorce, które daje nam projektowanie zorientowane na domenę, i nauczyć się, jak je wykorzystać podczas refaktoryzacji systemu.

Dlatego w tym rozdziale omówimy następujące główne tematy:

- jasne definiowanie pojęć,
- podział domeny,
- radzenie sobie z komunikacją między kontekstami ograniczonymi.

Po przeczytaniu tego rozdziału będziesz wiedział, jak strategicznie podzielić domenę na poddomeny, zrozumieć ich role w Twojej domenie i określić optymalne wzorce komunikacji. Ta operacja refaktoryzacji zaowocuje modułowym rozwiązaniem, które jest skalowalne i łatwe w utrzymaniu.

Zrozumiała definicja terminów rozwiązuje połowę problemu

W swojej książce Evans podkreśla znaczenie używania języka wszechobecnego w rozmowach z ekspertami dziedzinowymi i wszystkimi członkami zespołu. Aby zapewnić jasne zrozumienie problemu i uniknąć niejednoznaczności wynikających z nieprawidłowego tłumaczenia z języka domeny na język techniczny, ważne jest, aby język używany w naszej bazie kodu był jak najbardziej zbliżony do modelu domeny. Wynika to z faktu, że produkt, który wdrażamy na produkcji, wykorzystuje język techniczny, a nie biznesowy.

Aby to osiągnąć, podczas rozmów między członkami zespołu nikt nie musi tłumaczyć terminów z jednego języka na drugi. Ponieważ oprogramowanie nie radzi sobie z niejednoznacznością, powinniśmy opierać rozmowę na modelu domeny. Koncepcję modelu domeny dokładniej wyjaśnimy później w tym rozdziale, w punkcie „Czym jest kontekst ograniczony?”.

Evans jest w tej kwestii bezkompromisowy:

Dzięki wszechobecnemu stosowaniu języka opartego na modelu i jego ciągłemu udoskonalaniu dojdziemy do kompletnego i wyczerpującego modelu, stworzonego z prostych elementów, które w połączeniu ze sobą wyrażać będą złożone pojęcia. (...)

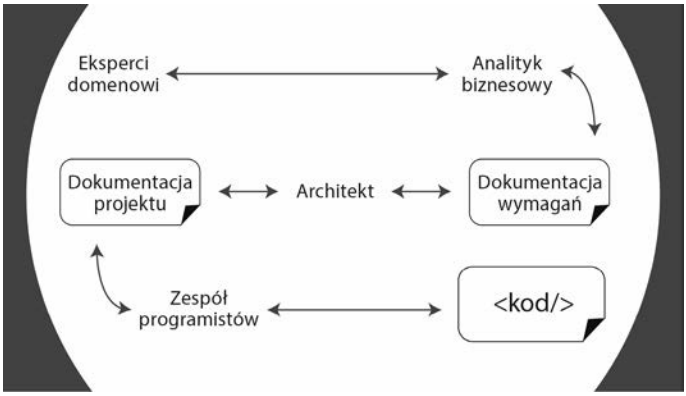
(...) Eksperci dziedzinowi powinni sprzeciwiać się pojęciom lub strukturom brzmiącym dziwnie lub nieadekwatnie do opisu znaczenia dziedziny. Natomiast programiści powinni doszukiwać się niejasności i niespójności, które mogłyby wypaczyć projekt.

Język wszechobecny to nie tylko słownictwo; to rygorystyczny język używany przez wszystkich członków zespołu, mający na celu usunięcie wszelkich niejednoznaczności i stworzenie podstawy do identyfikacji granic wokół problemu biznesowego, który ten zespół ma rozwiązać.

Podczas refaktoryzacji ważne jest zrozumienie modelu leżącego u podstaw oprogramowania, nad którym pracujemy, zanim zaczniemy modyfikować kod. Posiadanie wspólnego modelu biznesowego, który każdy członek zespołu może zrozumieć, jest kluczem do uzyskania przejrzystej architektury rozwiązania. Pomaga to jasno określić granice wokół problemu biznesowego, a posiadanie wspólnego języka jest pierwszym krokiem. Spróbuj pomyśleć o modelu klienta. Wydaje się, że stworzenie modelu naszego klienta, uwzględniającego wszystkie właściwości, takie jak dane do fakturowania, dane osobowe, adresy dostawy, adresy siedziby itp., jest łatwe. W ten sposób mielibyśmy kompletny model naszego klienta. Większość oprogramowania, nad którym pracujesz, prawdopodobnie ma podobny model i to stanowi główny problem, gdy musisz zmodyfikować zachowanie programu lub zaimplementować nową funkcję.

Jeśli musisz zaimplementować nową funkcję, na przykład w obszarze logistyki Twojej aplikacji, prawdopodobnie będziesz musiał zmodyfikować model klienta. Do tej pory zwykle byłeś zmuszony do zmiany modelu w obszarze sprzedaży i zakupów. Jak już wiesz, zmiana kodu w wielu obszarach jest jedną z głównych obaw programisty, ponieważ niemożliwe jest przewidzenie, co stanie się w systemie, gdy zmodyfikujesz wiele komponentów w swojej bazie kodu. Teraz powinno stawać się jasne, jak ważne jest posiadanie modelu dostosowanego do Twojego problemu biznesowego. Model, który jest zbyt duży, zmusza Cię do zmieniania go za każdym razem, gdy musisz zmodyfikować swój kod. Model, który jest zbyt mały, nie wystarczy do opisania samego problemu. Jak zaprojektować odpowiedni model? Gdy byłeś dzieckiem, prawdopodobnie bawiłeś się w głuchy telefon. Zabawne było odkrycie, że zdanie na końcu było zupełnie inne niż to na początku. Na każdym etapie, gdy dziecko szeptało zdanie do ucha następnej osoby w rzędzie, zdanie to nieuchronnie się zmieniało, co prowadziło do zabawnych rezultatów.

Rysunek 3.1 pokazuje wzorzec gry w głuchy telefon w rzeczywistej sytuacji biznesowej. Ekspert domenowy wyjaśnia swój problem analitykowi, który wyjaśnia ten sam (czy na pewno?) problem architektowi i tak dalej, aż to wyjaśnienie dociera do zespołu programistów. Oni tworzą rozwiązanie, które zostanie wdrożone. Czy widzisz problem? Każda osoba zaangażowana w ten wzorzec ma własną interpretację problemu, która różni się od interpretacji poprzedniej osoby w łańcuchu komunikacji.



Rysunek 3.1. Głuchy telefon w sytuacji biznesowej

Choć podczas zabawy efekt końcowy może być zabawny, w świecie biznesu to nie jest już tak śmieszne. Za każdym razem, gdy osoby ze strony biznesu próbują wyjaśnić problem specjalistom technicznym, używają własnego języka. Dla ludzi biznesu ich język jest jednoznaczny. Ponieważ znają problem i codziennie rozmawiają o nim z kolegami lub klientami, zakładają, że takie samo wzajemne zrozumienie istnieje również z innymi.

Doprowadziło to do stworzenia solidnego języka do komunikowania potrzeb i rozwiązań między tymi dwiema grupami. Specjaliści techniczni również mają swój własny język, który jest zupełnie inny od języka biznesu. Kiedy więc ci ludzie muszą stworzyć model rozwiązujący problem biznesowy opisany przez ludzi biznesu, muszą przetłumaczyć jeden język na drugi. Za każdym razem, gdy coś tłumaczysz, możesz stracić ważne informacje lub, co gorsza, możesz być zmuszony użyć własnych słów do opisania tego, co zrozumiałeś.

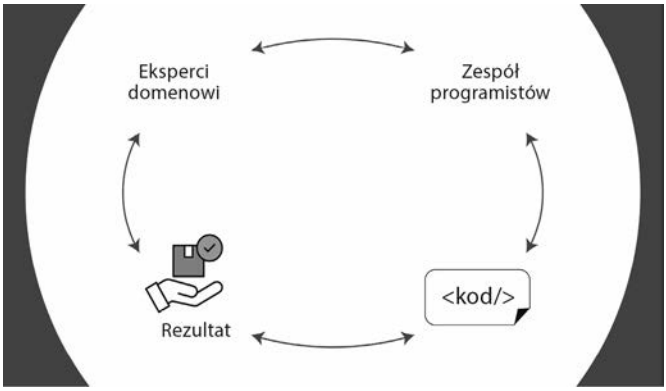
Ten proces powtarza się na każdym etapie, od klienta przez analityka po programistę. Pamiętasz, co mówiliśmy o złożoności, uprzedzeniach i tak dalej w poprzednim rozdziale?

Jak można uniknąć tego błędu? Pierwszym i najprostszym rozwiązaniem mogłoby być usunięcie granicy między ekspertami biznesowymi a programistami i doprowadzenie do bezpośredniej komunikacji między tymi ludźmi. Niestety, używają oni różnych języków i taki rodzaj komunikacji jest niemożliwy.

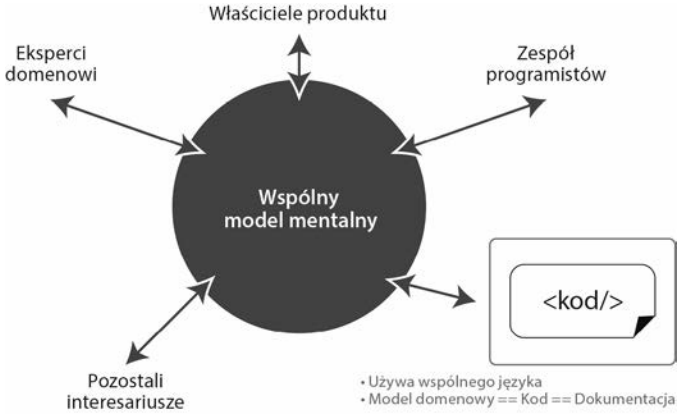
Rysunek 3.2 ilustruje to rozwiązanie. Eksperci domenowi i zespół programistów rozmawiają ze sobą bez pośrednika. W tym przypadku problemem nie jest utrata informacji, ale język. Każda z tych grup używa innego języka i nie może zrozumieć, co mówi druga strona.

Mam nadzieję, że dostrzegasz już znaczenie języka wszechobecnego w zespole. Gwarantuje on brak niejednoznaczności w komunikacji i stanowi pierwszy krok do stworzenia modelu domeny, który jest naprawdę przydatny dla wszystkich.

Wreszcie na rysunku 3.3 możesz zobaczyć rozwiązanie problemu. Dzięki zastosowaniu języka wszechobecnego każdy członek zespołu rozumie pozostałych, a wspólnie mogą stworzyć model domeny pomocny dla wszystkich.



Rysunek 3.2. Bezpośrednia komunikacja między zespołem biznesowym a technicznym



Rysunek 3.3. Wspólny model domeny

Posiadanie modelu zgodnego z biznesem znacznie ułatwia refaktoryzację, ponieważ Ty i eksperci biznesowi możecie się wzajemnie zrozumieć bez potrzeby tłumaczeń czy pośredników. Każde nowe wymaganie będzie idealnie dopasowane do Twojego modelu. Co więcej, jeśli nowe wymaganie wiąże się ze zmianą w modelu, możesz być spokojny, że nie wpłynie ona na pozostałe funkcjonalności systemu.

Dlaczego jednak potrzebujemy wielu modeli, a nie tylko jednego dla całej firmy? Jeśli naszym celem jest tworzenie oprogramowania, które może się rozwijać, musimy stworzyć wiele małych modeli, z których każdy jest odizolowany od pozostałych. To właśnie jest zadaniem kontekstów ograniczonych.

Czym jest kontekst ograniczony?

Kontekst ograniczony jest podstawowym wzorcem w projektowaniu domenowym. Jest on kluczowy dla wzorców strategicznych i stanowi centralną koncepcję przy pracy z dużymi modelami, ponieważ pozwala podzielić duży problem na mniejsze problemy lub duży model na mniejsze modele.

Posiadanie małych modeli pomaga nam rozwiązywać problemy biznesowe i utrzymywać nasz kod w przyszłości. Aby zidentyfikować granice wokół tego szczegółowego modelu, należy wykorzystać wcześniej odkryty język wszechobecny do identyfikacji kluczowych zdarzeń, które przenoszą dyskusję wokół problemu biznesowego z jednego kontekstu do drugiego.

W naszej podróży odkrywania wzorców w projektowaniu domenowym i ich implementacji wykorzystamy system ERP do zarządzania browarem. W tym przykładzie, jak zobaczysz, zaczniemy od warsztatu EventStormingu. Spotkasz się ze **zdarzeniem domenowym**, wzorcem, który omówimy w rozdziale 4., „Wzorce taktyczne". Na razie musisz wiedzieć, że istnieją specjalne zdarzenia, tzw. zdarzenia kluczowe, mające szczególne właściwości, które pomagają nam zidentyfikować te granice.

Jednym z takich zdarzeń jest utworzenie zamówienia sprzedaży — `SalesOrderCreated`. Jest to zdarzenie, które zmienia przebieg procesu. Na przykład po utworzeniu zamówienia sprzedaży musisz zweryfikować limit kredytowy klienta, dostępność piwa w magazynie oraz warunki wysyłki w obszarze logistyki. Reguły gry się zmieniły! Po tym zdarzeniu prawdopodobnie potrzebujesz nowych zasad do obsługi wszystkich tych przypadków. Te zasady mogą znajdować się w innym modelu, który może być również w różnych kontekstach ograniczonych.

Należy pamiętać, że istotą DDD jest tworzenie oprogramowania w oparciu o model, który jest zrozumiały dla wszystkich członków zespołu. Gdy ten model przestanie odzwierciedlać przepływ biznesowy ze względu na zwiększoną złożoność i zaangażowane w niego komponenty, zdecydowanie wykraczasz poza granice modelu. Przy refaktoryzacji starego projektu ważne jest zidentyfikowanie tych granic, ponieważ pierwszą rzeczą, którą musisz zrobić po ich odkryciu, jest usunięcie powiązań między obiektami należącymi do różnych kontekstów ograniczonych. Usunięcie tych powiązań umożliwia modyfikację części systemu bez obawy o uszkodzenie innych, co jest fundamentalnym krokiem!

Innym elementem, który możesz wykorzystać do wyznaczenia granic między kontekstami, są ludzkie zachowania. Wróćmy do przykładu utworzonego zamówienia sprzedaży: zachowanie osób zaangażowanych w system płatności będzie inne niż osób zaangażowanych w logistykę, ponieważ będą miały do wykonania różne zadania. Musisz uchwycić te różnice i wykorzystać je do identyfikacji granic swoich kontekstów.

Po podzieleniu całej domeny na wiele różnych kontekstów ograniczonych musisz znaleźć sposób, w jaki te konteksty komunikują się ze sobą. Podobnie jak w rzeczywistym świecie, każda część systemu przyczynia się do końcowego sukcesu, a jedynym sposobem na osiągnięcie celu jest *współpraca*.

Podział domeny na sensowne granice

Najbardziej oczywistą zaletą posiadania języka wszechobecnego jest możliwość łatwego identyfikowania granic wokół problemu biznesowego i podziału domeny na wiele poddomen. Domena biznesowa to główny obszar działalności firmy. W przykładzie, którego użyjemy w tej książce, domeną biznesową jest *zarządzanie browarem*. Usługa, którą świadczy nasza hipotetyczna firma, to produkcja i sprzedaż najlepszego piwa na świecie! Proste.

Ważne jest, aby zrozumieć, że domena biznesowa nie jest aplikacją komputerową. Oprogramowanie dostarcza rozwiązanie, które tworzysz, aby pomóc klientowi w zarządzaniu jego biznesem. Domena biznesowa jest zazwyczaj duża i złożona. Jak już wiesz, stworzenie jednego modelu opisującego cały biznes jest niemożliwe i prawdopodobnie jest to sytuacja, w której się znajdziesz, podchodząc do refaktoryzacji. Możesz zauważyć problem z posiadaniem „uniwersalnego" modelu. Jest to ograniczenie dla każdego nowego wymagania i tworzy obawę przed zmianą czegokolwiek w kodzie ze względu na silne powiązania między komponentami Twojej starej aplikacji.

Podobnie jak w rzeczywistości, firma osiąga swoje cele w domenie biznesowej, dzieląc sam biznes na wiele obszarów. DDD robi to samo, dzieląc domenę na wiele poddomen. Poddomena to mała część całej domeny, skupiająca się na konkretnym aspekcie biznesu. Jest to ważny wzorzec strategiczny w ramach DDD. Dlatego jedna poddomena nie wystarczy, aby zagwarantować sukces firmy. Jest to trybik w bardziej złożonym systemie — domenie.

W złożonym systemie każde koło zębate jest zamontowane tak, aby cały system mógł spełnić swoje zadanie. Nie każde koło zębate jest równie ważne. Jeśli złamiesz szprychę w rowerze, możesz kontynuować pedałowanie i dojechać do celu. Jeżeli jednak przebijesz oponę, musisz przerwać jazdę, naprawić oponę i dopiero wtedy możesz dokończyć trasę. Zarówno opona, jak i szprycha są częściami koła, ale opona jest ważniejsza. Podobnie jeśli celem Twojej firmy jest produkcja najlepszego piwa, posiadanie dobrego systemu magazynowego będzie ważne, by przechowywać piwa przed sprzedażą, ale mniej istotne niż posiadanie doskonałego działu produkcji. Żaden z tych działów sam w sobie nie sprawi, że Twój browar będzie dochodowy. Wszystkie te działy, i prawdopodobnie inne, są niezbędne do osiągnięcia celu, ale kluczowe jest zidentyfikowanie, za co każdy z nich jest odpowiedzialny.

Aby zidentyfikować granice wokół tych poddomen, musisz użyć języka wszechobecnego odkrytego w fazie eksploracji z ekspertami domenowymi i interesariuszami. Każda niespójność w języku wszechobecnym jest ostrzeżeniem, że coś wymaga dokładniejszego zbadania. Dzieje się tak, ponieważ możemy mówić o tych samych rzeczach, ale używając różnych nazw, lub możemy patrzeć na zupełnie inną rzecz, która może należeć do innego kontekstu ograniczonego. Musisz się nauczyć, jak zmuszać swoich ekspertów biznesowych do pozostawania w przestrzeni problemów, ponieważ wiesz, jak to jest ważne. Musisz określić, czy mówisz o kliencie w kontekście zamówienia sprzedaży, czy w kontekście dostawy. Z pewnością w obu przypadkach mówisz o kliencie, ale na dwa różne sposoby. Oznacza to posiadanie dwóch różnych modeli, a w tym przypadku dwóch różnych poddomen. Już wiesz, że poddomena to mniejszy obszar wiedzy lub działalności w ramach większej domeny. Teraz odkryłeś również, że — podobnie jak w przypadku zębatki lub koła — niektóre poddomeny są ważniejsze od innych i istnieją różne ich typy. Podział całej domeny na wiele poddomen oznacza, że możesz stworzyć mniejszy model i rozmawiać z konkretnymi ekspertami domenowymi, aby głębiej wejść w modelowanie. Podczas pierwszej części refaktoryzacji niezwykle ważne jest wydzielenie domeny, aby rozpocząć od dobrego projektu strategicznego. Proces refaktoryzacji ma na celu poprawę projektu, struktury i implementacji oprogramowania, a także jego *niefunkcjonalnych* cech bez zmiany funkcjonalności.

Twoim celem jest zmniejszenie złożoności przypadkowej i poprawa czytelności kodu bez efektów ubocznych. Pamiętaj, że nie możesz przekształcić dużej, skomplikowanej aplikacji w rozwiązanie oparte na mikrousługach w jednym kroku. Zawsze miej na uwadze prawo Galla:

Złożony system, który działa, to system, który rozwinął się z prostego systemu. I na odwrót: złożony system zaprojektowany od samego początku jako złożony i stworzony od razu w całości nigdy nie będzie działał i poprawianie go nic nie da. Trzeba go stworzyć ponownie jako system początkowo prosty.

Ogólnie rzecz biorąc, refaktoryzacja polega na wdrażaniu zestawu znormalizowanych modyfikacji na małą skalę, gdzie każda zmiana jest drobną korektą kodu źródłowego programu komputerowego. Te poprawki zazwyczaj zachowują istniejącą funkcjonalność oprogramowania, a przynajmniej nie zmieniają jego zgodności ze specyfikacjami funkcjonalnymi. Autor książki *Refaktoryzacja do wzorców projektowych*, Joshua Kerievsky, napisał:

Ciągłe udoskonalanie struktury kodu sprawia, że praca z nim staje się coraz łatwiejsza. Jest to zupełne przeciwieństwo typowego podejścia, w którym przeprowadza się niewiele refaktoryzacji, a skupia się głównie na szybkim dodawaniu nowych funkcji. Jeśli wyrobisz w sobie nawyk ciągłej refaktoryzacji, przekonasz się, że rozbudowa i utrzymanie kodu stają się znacznie prostsze.

Pierwszym krokiem tego procesu jest podział całej domeny na wiele poddomen, co nazywamy **mapowaniem kontekstu**.

Należy podkreślić, że podział domeny na wiele poddomen niekoniecznie wymaga tworzenia wielu autonomicznych usług. Implementacja może przybrać formę modularnego monolitu lub rozwiązania opartego na mikrousługach. Kluczowe jest to, by kontekst ograniczony pozostawał samodzielny i nie rozciągał się na wiele usług lub modułów.

Mapowanie kontekstu

Mapowanie kontekstu to technika umożliwiająca zwizualizowanie i zrozumienie relacji oraz interakcji między kontekstami ograniczonymi w systemie. Polega na stworzeniu mapy, która pokazuje, jak konteksty są ze sobą powiązane, identyfikuje punkty integracji i ujawnia potencjalne problemy. Taka mapa staje się kluczowym narzędziem w zarządzaniu złożonością dużych systemów.

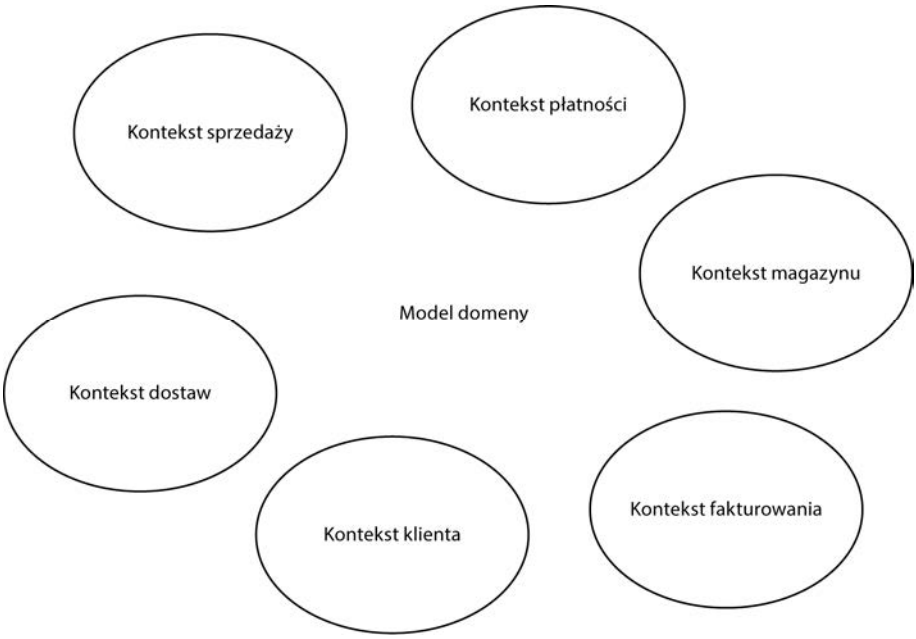
Stworzenie takiej mapy nie jest jednak łatwym zadaniem. Należy wykonać szereg kroków w pętli, aż Ty i Twój zespół osiągniecie porozumienie w kwestii wyniku.

Pierwszym krokiem jest zidentyfikowanie kontekstów ograniczonych w swoim systemie. Każdy kontekst powinien mieć wyraźne granice i model domeny. Gdy już określisz granice, możesz przejść do następnego kroku, który początkowo może wydawać się trudny. Polega on na zdefiniowaniu, jak te konteksty oddziałują na siebie. Czy są od siebie zależne? Czy dzielą dane lub usługi?

Gdy uporasz się z tym zadaniem i stworzysz zrozumiałą sieć połączeń, powinieneś spróbować wyróżnić punkty, w których konteksty integrują się lub muszą się komunikować.

Na koniec powinieneś być w stanie narysować wizualną reprezentację kontekstów i ich relacji. Pod koniec tego rozdziału otrzymasz kompletny schemat ilustrujący przepływ informacji i zależności, jak pokazano na rysunku 3.12. Zanim jednak do tego przejdziemy, rozłóżmy to na mniejsze kroki, aby w pełni zrozumieć mapowanie kontekstu.

Na początek, rysunek 3.4 pokazuje, jak moglibyśmy narysować mapę kontekstu naszego systemu ERP, używając kontekstów ograniczonych dla klienta, sprzedaży, magazynu i wysyłki.



Rysunek 3.4. Fragment mapy kontekstów naszego browaru

Rodzaje poddomen

W naszej liście zadań mających na celu stworzenie dobrego mapowania pierwszym krokiem jest zdefiniowanie konkretnych modeli domenowych. Co to jednak oznacza w praktyce?

Oznacza to, że w architekturze systemu każdy kontekst ograniczony pełni określoną rolę, a Twoim zadaniem jest określenie wagi każdego z nich. Wiąże się to z identyfikacją, które konteksty ograniczone reprezentują główną logikę biznesową, które ją wspierają, a które mają bardziej ogólny charakter. Dzięki temu możesz nadawać odpowiedni priorytet swoim wysiłkom i zasobom.

W tym celu musisz przeanalizować każdy kontekst ograniczony i przypisać go do jednego z typów poddomen. Ta klasyfikacja pomaga zrozumieć strategiczne znaczenie każdej części systemu. Poddomeny reprezentują konkretne obszary w ramach domeny. Odpowiadają one poszczególnym kontekstom ograniczonym, z własnymi zestawami reguł i logiki.

W mapowaniu kontekstów możesz wybrać spośród kilku typów poddziedzin, w zależności od ich roli w systemie:

- poddomena główna,
- poddomena pomocnicza,
- poddomena ogólna.

Poprzez identyfikację typu każdego z Twoich kontekstów ograniczonych możesz podejmować świadome decyzje dotyczące podejścia do ich rozwoju, integracji i utrzymania. Na przykład prawdopodobnie zainwestujesz więcej zasobów i innowacji w domenę główną, natomiast dla domen ogólnych możesz rozważyć zastosowanie gotowych rozwiązań.

Ten proces identyfikacji i kategoryzacji jest kluczowy, ponieważ stanowi fundament mapy kontekstów, kierując ogólną strategią architektoniczną i pomagając dostosować wysiłki ze strony zespołu technicznego do priorytetów biznesowych. Ma to również silny wpływ na strukturę i skład zespołów. Na przykład zespół pracujący nad domeną główną będzie prawdopodobnie początkowo składał się z bardziej doświadczonych programistów, a później niektórzy z nich mogą zostać przeniesieni do innych zespołów w celu dzielenia się wiedzą.

Teraz, gdy rozumiesz już mapy kontekstów, przyjrzyjmy się dokładnie, czym one są i jakim celom służą.

Poddomena główna

Poddomena główna stanowi serce biznesu. To właśnie tutaj znajduje się unikalna propozycja wartości Twojej firmy i tu masz przewagę konkurencyjną. Jest to najbardziej kluczowa i złożona część Twojego systemu, wymagająca największej uwagi i zasobów. W przypadku naszego systemu ERP domena główna to system sprzedaży, ponieważ jest kluczowy dla działalności i sukcesu firmy.

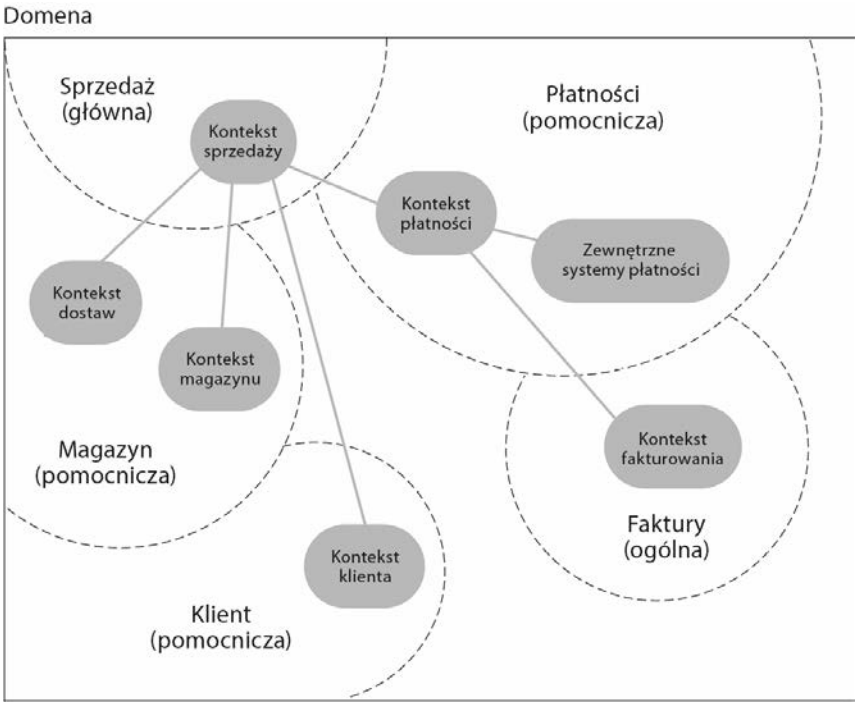
Poddomena pomocnicza

Poddomeny pomocnicze, choć ważne, nie są głównym obszarem zainteresowania firmy. Uzupełniają poddomenę główną i umożliwiają jej działanie, ale nie zapewniają przewagi konkurencyjnej. Te poddomeny są niezbędne do sprawnego funkcjonowania firmy, ale nie są tym, co wyróżnia ją na tle konkurencji. W naszym przykładzie ERP poddomeną pomocniczą byłby moduł obsługi klienta — jest kluczowy dla zadowolenia klientów, ale nie stanowi głównego motora biznesu.

Poddomena ogólna

Poddomeny ogólne są powszechne w wielu firmach i nie są charakterystyczne dla konkretnego przedsiębiorstwa. Często są to obszary, w których nie ma potrzeby wymyślania koła na nowo i można skorzystać z istniejących rozwiązań lub zlecić je na zewnątrz. Dobrym przykładem poddomeny ogólnej jest system fakturowania — jest niezbędny do monitorowania finansów, ale nie jest obszarem, na którym browar skupiałby swoje innowacyjne wysiłki.

Rysunek 3.5 przedstawia mapę kontekstów z definicją każdej poddomeny.



Rysunek 3.5. Rodzaje poddomen

Zrozumienie różnych typów poddomen jest kluczowe dla efektywnego mapowania kontekstu i ogólnego projektowania systemu. Pomaga w ustalaniu priorytetów prac rozwojowych, efektywnym przydzielaniu zasobów oraz podejmowaniu strategicznych decyzji dotyczących integracji i rozwoju systemu. Dzięki rozpoznaniu, które części systemu należą do poszczególnych kategorii poddomen, można skupić wysiłki innowacyjne tam, gdzie mają one największe znaczenie, jednocześnie znajdując wydajne rozwiązania dla mniej krytycznych obszarów.

Poddziedziny zmieniają się w czasie

To, co zaprojektowaliśmy w tym rozdziale, to pierwsza mapa kontekstów, oparta na wiedzy zdobytej podczas rozmów z ekspertami biznesowymi browaru. Staraliśmy się zaspokoić rzeczywiste potrzeby biznesowe. Z czasem nasza wiedza o domenie będzie się pogłębiać, a procesy biznesowe będą się zmieniać — z inicjatywy ekspertów biznesowych lub bez niej. W związku z tym nasze konteksty ograniczone lub ich znaczenie w domenie mogą ulec zmianie. Co jest równie ważne, jeśli była rotacja zespołu, każdy członek powinien być w stanie zająć się prawie każdą poddomeną.

Zarządzanie komunikacją między kontekstami ograniczonymi

Teraz, gdy już dobrze rozumiesz znaczenie odpowiednio zdefiniowanych granic, możemy przejść do ostatniego aspektu kontekstów ograniczonych — komunikacji. Aby zbudować spójny system, te konteksty ograniczone muszą skutecznie ze sobą współdziałać. Dlatego właśnie kluczowa jest odpowiednia komunikacja między kontekstami ograniczonymi. Projektowanie strategiczne oferuje pewne wzorce, które pomogą Ci zidentyfikować najlepsze rozwiązania w oparciu o kontekst działania. Twoim celem jest uzyskanie jasnego obrazu tego, jak różne części systemu współdziałają i odnoszą się do siebie nawzajem.

Znaczenie prawidłowej komunikacji

Skuteczna komunikacja między kontekstami ograniczonymi sprawia, że system działa jako spójna całość. Gdy konteksty ograniczone nie komunikują się prawidłowo, pojawiają się niespójności i problemy z integracją, prowadząc do fragmentacji systemu. Odpowiednie mechanizmy komunikacji pomagają w następujący sposób:

- **Utrzymanie spójności** sprawia, że dane i zachowania są spójne w całym systemie.
- **Zwiększenie elastyczności** umożliwia niezależny rozwój kontekstów ograniczonych bez naruszania integralności systemu.
- **Zmniejszenie powiązań** minimalizuje zależności między różnymi częściami systemu, ułatwiając zarządzanie i refaktoryzację.
- **Poprawa przejrzystości** zapewnia jasne wzorce interakcji, co pomaga w rozumieniu i utrzymywaniu systemu.

Wzorce komunikacji

Te wzorce strategiczne to wysokopoziomowe strategie projektowe stosowane do zarządzania relacjami między kontekstami ograniczonymi. Zapewniają one schemat postępowania w kwestii integracji, współpracy i zarządzania pomiędzy kontekstami. Do najważniejszych wzorców można zaliczyć między innymi następujące:

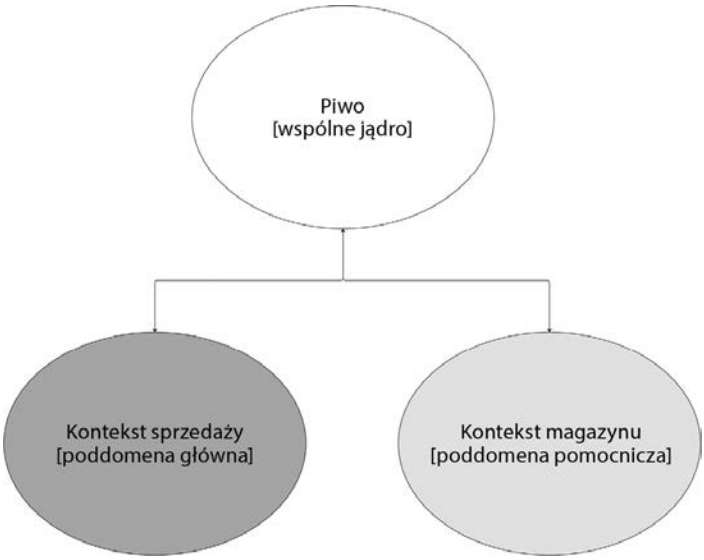
- **Wspólne jądro**,
- **Klient-dostawca**,
- **Konformista**,
- **Warstwa zapobiegająca uszkodzeniu**,
- **Usługa otwartego hosta**,
- **Język opublikowany**,
- **Oddzielne drogi**.

Wspólne jądro

We wzorcu wspólnego jądra dwa lub więcej kontekstów ograniczonych dzielą część tego samego modelu domeny. Ta współdzielona część jest zarządzana wspólnie przez zespoły odpowiedzialne za każdy kontekst. Ten wzorzec jest przydatny, gdy konteksty są silnie współzależne i wymagają spójnego zrozumienia określonych aspektów modelu.

Używając naszego systemu ERP jako przykładu, przyjrzyjmy się kontekstom magazynu i sprzedaży, które dzielą informacje o piwie i poziomach zapasów. Dzięki temu wspólnemu jądru oba konteksty mają takie samo zrozumienie danych o produktach, co zmniejsza ryzyko rozbieżności. Kontekst magazynu zarządza poziomami zapasów, szczegółami dotyczącymi piwa i miejscem składowania w magazynie, natomiast kontekst sprzedaży obsługuje zamówienia klientów, przetwarzanie zamówień i realizację.

Wspólne jądro w tym scenariuszu obejmuje szczegóły dotyczące piwa (patrz rysunek 3.6), takie jak jednostka magazynowa (**SKU**, ang. *stock keeping unit*), opis i cena, które są kluczowe dla prawidłowego funkcjonowania obu kontekstów.



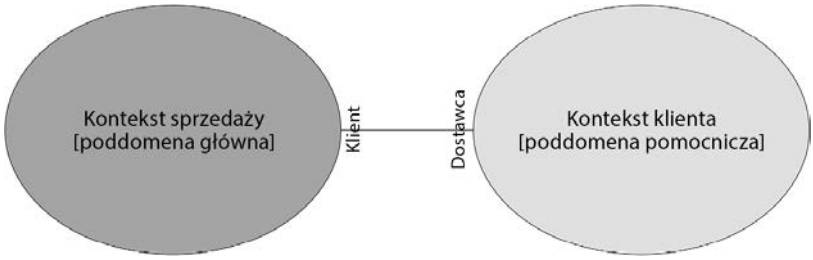
Rysunek 3.6. Przykład wspólnego jądra

Klient-dostawca

Wzorzec Klient-dostawca ustanawia wyraźną relację góra – dół między kontekstami. Kontekst znajdujący się wyżej w hierarchii dostarcza usługi lub dane, a kontekst niżej w hierarchii jest od nich zależny. Dostawca (wyżej) musi sprawić, by klient (niżej) mógł polegać na jego usługach, co sprzyja współpracy.

Na przykład kontekst klienta (dostawca) dostarcza informacje o klientach do kontekstu sprzedaży (klient), jak widać na rysunku 3.7. Kontekst sprzedaży polega na dokładnych danych klientów, aby skutecznie zarządzać zamówieniami. Kontekst sprzedaży zajmuje się obsługą zamówień i przetwarzaniem płatności, a kontekst klienta zarządza limitami

kredytowymi i specjalnymi kategoriami. Kontekst klienta musi sprawić, by informacje o klientach były dokładne i dostępne na czas, aby kontekst sprzedaży mógł działać prawidłowo.

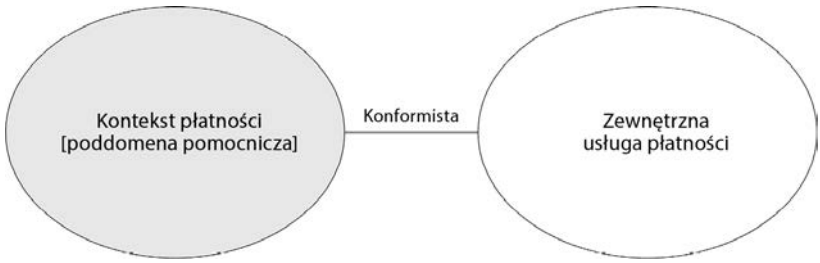


Rysunek 3.7. Przykład wzorca Klient-dostawca

Konformista

W modelu konformistycznym kontekst działający w dole strumienia przetwarzania (konformista) przyjmuje model i protokoły komunikacyjne kontekstu z góry strumienia bez wpływania na niego. Ten wzorzec jest stosowany, gdy kontekst z dołu strumienia ma niewielką kontrolę nad kontekstem z góry i musi dostosować się do jego sposobu działania.

Na przykład zewnętrzna usługa przetwarzania płatności narzuca format danych i protokoły interakcji, a wewnętrzny kontekst płatności musi się do tych wymagań dostosować (patrz rysunek 3.8). Usługa przetwarzania płatności (zewnętrzna) zapewnia bramki płatnicze, przetwarzanie transakcji i wykrywanie oszustw, natomiast wewnętrzny kontekst płatności zarządza wewnętrznymi przepływami płatności, danymi płatniczymi użytkowników i rejestrami transakcji. Wewnętrzny kontekst płatności musi dostosować się do protokołów i formatów danych określonych przez zewnętrzną usługę przetwarzania płatności, aby zapewnić płynną integrację.

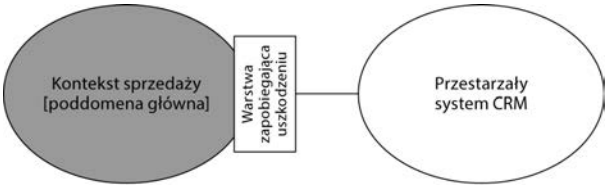


Rysunek 3.8. Przykład wzorca Konformista

Warstwa zapobiegająca uszkodzeniu

Wzorzec warstwy zapobiegającej uszkodzeniu (**ACL**, ang. *anti-corruption layer*) wprowadza warstwę, która tłumaczy i dostosowuje modele dwóch kontekstów ograniczonych. Warstwa ta zapobiega zanieczyszczeniu kontekstu docelowego przez model kontekstu źródłowego, pozwalając każdemu z nich zachować integralność.

W naszej platformie ERP kontekst sprzedaży może wykorzystywać warstwę zapobiegającą uszkodzeniu do tłumaczenia danych z przestarzałego systemu CRM na swój własny model. Dzięki temu kontekst sprzedaży nie jest zanieczyszczony przez przestarzały model CRM. Stary system CRM zarządza relacjami z klientami, danymi historycznymi i interakcjami sprzedażowymi, a kontekst sprzedaży obsługuje bieżące procesy sprzedażowe, interakcje z klientami i zarządzanie zamówieniami. Warstwa zapobiegająca uszkodzeniu tłumaczy dane CRM na format i strukturę wymagane przez kontekst sprzedaży, zapewniając integralność i spójność danych (patrz rysunek 3.9).



Rysunek 3.9. Przykład warstwy zapobiegającej uszkodzeniu

Usługa otwartego hosta

Wzorzec usługi otwartego hosta (**OHS**, ang. *open host service*) polega na udostępnianiu możliwości kontekstu ograniczonego poprzez dobrze zdefiniowany interfejs usługi, czyniąc go otwartym hostem dla interakcji z innymi kontekstami. Wzorzec ten promuje luźne powiązania i jasne kontrakty między kontekstami.

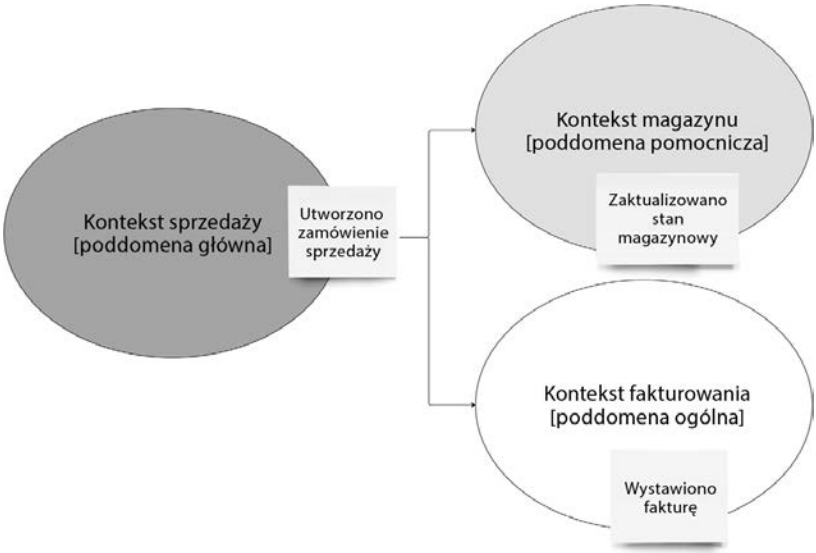
Na przykład kontekst dostawy może udostępniać wzorzec usługi otwartego hosta, który pozwala innym kontekstom, takim jak sprzedaż i magazyn, na żądanie usług wysyłkowych poprzez standardowe API. Kontekst dostawy zarządza stawkami wysyłkowymi, przewoźnikami i informacjami o śledzeniu przesyłek. Kontekst sprzedaży żąda usług dostawy do realizacji zamówień, a kontekst magazynu żąda informacji o wysyłce do przewozów i uzupełnień zapasów. Kontekst dostawy udostępnia swoje usługi poprzez API, umożliwiając innym kontekstom interakcję bez ścisłego powiązania (patrz rysunek 3.10).

Rysunek 3.10. Przykład usługi otwartego hosta

Język opublikowany

Wzorzec języka opublikowanego polega na zdefiniowaniu wspólnego języka (zestawu wspólnych terminów i struktur danych), którego wiele kontekstów ograniczonych używa do komunikacji. Ten wspólny język zmniejsza prawdopodobieństwo wystąpienia nieporozumień i upraszcza integrację.

W architekturze mikrousług do komunikacji między usługami często stosuje się wspólny schemat zdarzeń. Dzięki temu wszystkie usługi spójnie interpretują dane zdarzeń. Na przykład kontekst sprzedaży publikuje zdarzenia zamówień, gdy składane są nowe zamówienia. Kontekst magazynu odbiera zdarzenia zamówień, aby aktualizować stany magazynowe, a kontekst fakturowania odbiera zdarzenia zamówień, aby generować faktury. Język opublikowany definiuje wspólny schemat dla zdarzeń zamówień, co gwarantuje, że wszystkie usługi mogą poprawnie interpretować dane (patrz rysunek 3.11).



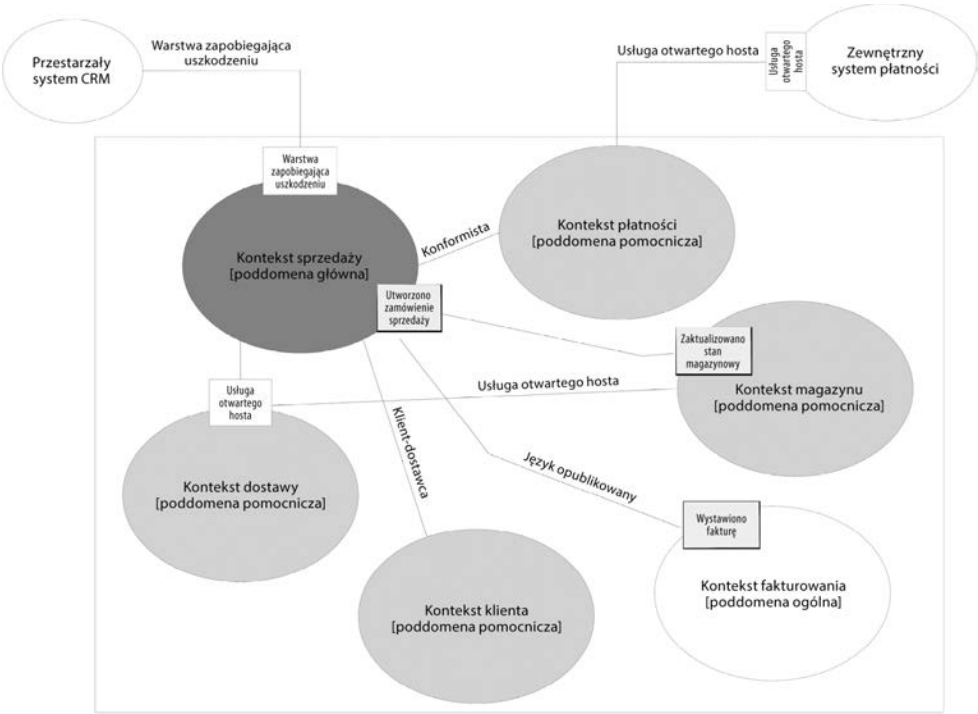
Rysunek 3.11. Przykład opublikowanego języka

Oddzielne drogi

Wzorzec oddzielnych dróg rozwiązuje problem złożoności, która pojawia się, gdy dwie odrębne części systemu muszą się rozwijać niezależnie. Wzorzec ten polega na identyfikacji w ramach domeny kontekstów ograniczonych, które mają minimalną interakcję i mogą być rozwijane oraz utrzymywane oddzielnie. Poprzez wyraźne rozdzielenie tych kontekstów, zespoły mogą skupić się na konkretnych potrzebach i regułach każdego obszaru domeny, nie będąc hamowanymi przez zawiłości innych obszarów. To rozdzielenie zmniejsza sprzężenie i umożliwia bardziej elastyczne i responsywne cykle rozwoju, pozwalając każdemu kontekstowi ewoluować we własnym tempie i w swoim kierunku.

Końcowa mapa kontekstu

Na rysunku 3.12 możesz zobaczyć pełną mapę kontekstów naszego systemu ERP, przedstawiającą wzorce komunikacji między poszczególnymi kontekstami ograniczonymi. Możesz zacząć od jednego wzorca, na przykład *konformistycznego*, a następnie przejść do *usługi otwartego hosta*, jeśli konteksty ograniczone muszą pozostać rozdzielone. Gdy musisz komunikować się z systemami zewnętrznymi, możesz zdecydować się na podejście *konformistyczne* lub użyć *warstwy zapobiegającej uszkodzeniu*, aby zachować swój język wewnątrz modelu domeny. Nie jesteś zmuszony do korzystania tylko z jednego rozwiązania — powinieneś używać odpowiedniego narzędzia w odpowiednim miejscu.



Rysunek 3.12. Pełna mapa kontekstu

Refaktoryzacja złożonych systemów z wykorzystaniem zasad DDD wymaga starannego przemyślenia sposobu komunikacji między kontekstami ograniczonymi. Odpowiednia komunikacja zapewnia spójność, elastyczność i przejrzystość, umożliwiając systemowi spójne funkcjonowanie. Dzięki wzorcom strategicznym, takim jak Wspólne jądro, Klient-dostawca, Konformista, Warstwa zapobiegająca uszkodzeniu, Usługa otwartego hosta czy Język opublikowany, zespoły mogą skutecznie zarządzać relacjami i interakcjami między kontekstami ograniczonymi.

Podsumowanie

W tym rozdziale zagłębiłeś się w strategiczne wzorce DDD, odkrywając, jak istotne jest ustanowienie języka wszechobecnego w celu wyeliminowania niejednoznaczności i zapewnienia zrozumiałej komunikacji między wszystkimi członkami zespołu.

Dowiedziałeś się, że dostosowanie języka używanego w kodzie do modeli domeny pomaga stworzyć przejrzystą i zrozumiałą architekturę, upraszczając przyszłe zmiany i zmniejszając liczbę błędów. Dowiedziałeś się, jak ważne jest dzielenie dużych modeli na mniejsze, łatwiejsze w zarządzaniu konteksty ograniczone, co jest kluczowe dla utrzymania systemu, który może rozwijać się bez obaw o nieprzewidziane konsekwencje. Takie podejście pozwala na izolację różnych obszarów biznesowych, ułatwiając modyfikację części systemu bez zakłócania innych. Na koniec poznałeś kilka strategicznych wzorców, które pomagają zarządzać relacjami i interakcjami między tymi kontekstami. Należą do nich wzorce: Wspólne jądro, Klient-dostawca, Konformista, Warstwa zapobiegająca uszkodzeniu, Usługa otwartego hosta i Język opublikowany.

Każdy z tych wzorców oferuje schemat utrzymania spójności, elastyczności i przejrzystości w całym systemie, zapewniając jego funkcjonowanie jako spójnej całości. Jeśli będziesz rozumiał te wzorce, to będziesz potrafił identyfikować granice w swojej domenie i skutecznie zarządzać złożonością dużych systemów.

W następnym rozdziale zagłębisz się w taktyczne wzorce projektowe, które dostarczą Ci praktycznych narzędzi i technik do wdrożenia poznanych dotąd koncepcji.

DDD: logika biznesowa to serce każdego dobrego projektu!

Współczesne aplikacje muszą łączyć niespotykaną wcześniej zwinność i skalowalność z precyzyjnym dopasowaniem do celów biznesowych. Od programistów i architektów wymaga się dziś nie tylko doskonałości technicznej, ale także zrozumienia zasad rządzących daną domeną. Tę filozofię znakomicie wspiera projektowanie zorientowane domenowo — *domain-driven design* (DDD).

Dzięki tej książce poznasz kluczowe koncepcje i praktyczne wskazówki, które pomogą Ci przekształcić monolit w nowoczesny, modułowy system. Autorzy szczegółowo omawiają strategiczne wzorce DDD, takie jak ograniczone konteksty i język wszechobecny, które poprawiają komunikację między ekspertami technicznymi i dziedzinowymi. Nauczysz się technik modelowania ułatwiających kontrolę nad złożonością i zwiększających elastyczność oprogramowania. Dowiesz się także, jak integrować mikrousługi, zarządzać komunikacją między nimi i zapewniać spójność danych i transakcji. W efekcie nauczysz się projektować systemy, w których logika biznesowa pozostaje w centrum wszystkich decyzji projektowych.

W książce między innymi:

- rozgraniczanie komponentów systemu
- wzorce strategiczne: ograniczone konteksty i język wszechobecny
- wzorce taktyczne do budowania agregatów i encji
- główne wzorce refaktoryzacji i ich wdrażanie
- architektury sterowane zdarzeniami a powiązania między komponentami
- testowanie integralności architektury

Alessandro Colla jest doświadczonym specjalistą do spraw handlu internetowego, programistą C# i ekspertem w dziedzinie projektowania zorientowanego na domenę. Zbudował od podstaw oprogramowanie zarządzające i systemy ERP dla różnych branż.

Alberto Acerbis jest architektem oprogramowania z wieloletnim doświadczeniem i szkoleniowcem. Jest też specjalistą w zakresie .NET, programistą C# i laureatem tytułu Microsoft MVP. Aktywnie uczestniczy w projektach *open source*.

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 978-83-289-3327-9
9 788328 933279
Cena: 89,00 zł
‹packt›