

O'REILLY®

Head First

Architektura oprogramowania

Rusz głową!

Przewodnik
po myśleniu
architektonicznym

Raju Gandhi
Mark Richards
Neal Ford



Helion

Tytuł oryginału: Head First Software Architecture: A Learner's Guide to Architectural Thinking

Tłumaczenie: Piotr Rajca

ISBN: 978-83-289-1567-1

© 2025 Helion S.A.

Authorized Polish translation of the English edition of *Head First Software Architecture*
ISBN 9781098134358 © 2024 Defmacro Software L.L.C., Mark Richards and Neal Ford.

This translation is published and sold by permission of O'Reilly Media, Inc.,
which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form
or by any means, electronic or mechanical, including photocopying, recording or
by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu
niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii
metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym,
magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi
bądź towarowymi ich właścicieli.

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<https://helion.pl/user/opinie/aropru>

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

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Spis treści (podsumowanie)

	Wprowadzenie	xxi
1	Architektura oprogramowania bez tajemnic. <i>Zaczynamy!</i>	1
2	Cechy architektury. <i>Znaj swoje właściwości</i>	39
3	Dwa prawa architektury oprogramowania. <i>Kompromisy... nic, tylko kompromisy!</i>	77
4	Komponenty logiczne. <i>Elementy konstrukcyjne</i>	117
5	Style architektury. <i>Kategoryzacja i filozofie</i>	159
6	Architektura warstwowa. <i>Separując zagadnienia</i>	175
7	Modularne monolity. <i>Bazujące na dziedzinie</i>	203
8	Architektura mikrojądra. <i>Konstruowanie dostosowań</i>	233
9	Zrób to sam. <i>Aplikacja podróżnicza LuzTravel</i>	261
10	Architektura mikrousług. <i>Kawałek po kawałku</i>	287
11	Architektura sterowana zdarzeniami. <i>Asynchroniczne przygody</i>	331
12	Zrób to sam. <i>Sprawdzanie swojej wiedzy</i>	383
	Dodatek A: pozostałości. <i>Sześć najważniejszych zagadnień, których nie opisaliśmy</i>	407
	Skorowidz	421

Spis treści (ten właściwy)

Wprowadzenie

Architektura oprogramowania jest trudnym zagadnieniem, więc Twój mózg będzie się starał oszukać Cię i przekonać, że nie jesteś w stanie się jej nauczyć. Będzie Ci podsuwał takie myśli: „Lepiej skoncentrować się na czymś ważnym, takim jak zjedzenie lanczu lub rozważania o tym, czy świny umieją latać”. Na szczęście to Ty MOŻESZ oszukać swój mózg i przekonać go, że opanowanie architektury oprogramowania jest niezwykle ważne, a ten rozdział pokaże Ci, jak to zrobić.

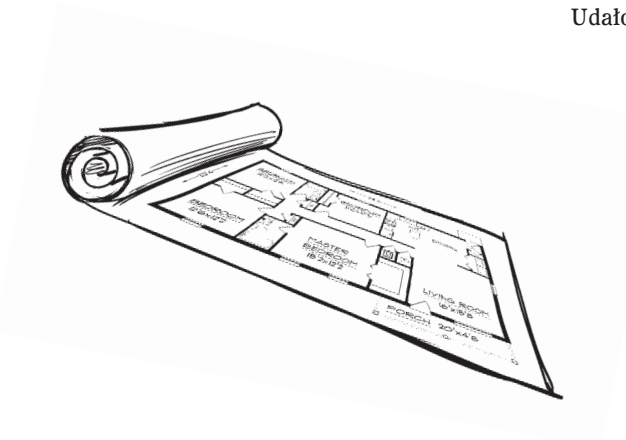
	Dla kogo jest ta książka?	xxii
	Wiemy, co myślisz	xxiii
	Wiemy, co myśli Twój mózg	xxiii
	Metapoznanie — myślenie o myśleniu	xxv
	Oto co MY zrobiliśmy	xxvi
	To, co TY możesz zrobić, aby zmusić swój mózg do posłuszeństwa	xxvii
	Przeczytaj to	xxviii
	Rozdziały „zrób to sam”	xxx
	Zespół recenzentów technicznych	xxxi
	Wspólne podziękowania	xxxii
	Indywidualne podziękowania	xxxiii

1

Architektura oprogramowania bez tajemnic
Zaczynamy!

Architektura oprogramowania ma fundamentalne znaczenie dla sukcesu systemu. Ten rozdział jest poświęcony omówieniu architektury oprogramowania. Czytając go, zrozumiesz, czym są wymiary architektury, i poznasz różnice pomiędzy architekturą oprogramowania a projektowaniem. Dlaczego jest to ważne? Ponieważ zrozumienie i stosowanie praktyk architektonicznych pomaga budować bardziej efektywne i poprawne oprogramowanie — oprogramowanie, które nie tylko lepiej funkcjonuje, ale także spełnia potrzeby i łagodzi obawy biznesu, a co więcej, zapewnia ciągłe poprawne działanie mimo bezustannych zmian, którym podlegają środowiska biznesowe i techniczne. Tak więc, bez dalszej zwłoki, zaczynamy.

Budowanie zrozumienia architektury oprogramowania	2
Plan budynku i architektura oprogramowania	3
Wymiary architektury oprogramowania	4
Rozgryzanie wymiarów	5
Pierwszy wymiar: cechy architektury	6
Drugi wymiar: decyzje architektoniczne	8
Trzeci wymiar: komponenty logiczne	10
Czwarty wymiar: style architektury	12
Z punktu widzenia projektu	16
Z punktu widzenia architektury	17
Spektrum pomiędzy architekturą a projektem	18
W którym miejscu spektrum wypada Twoja decyzja?	19
Strategiczna czy taktyczna	20
Wysoki czy niski nakład pracy	22
Kompromisy znaczące i mniej znaczące	24
Łączenie wszystkiego w całość	26
Udało Ci się!	27

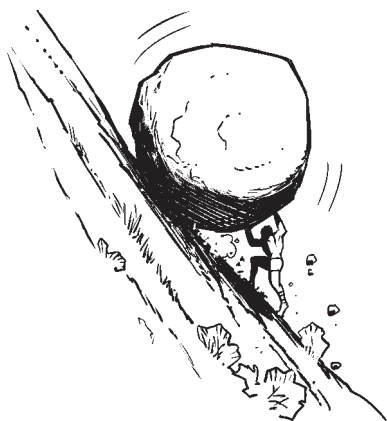


2

Cechy architektury

Znaj swoje właściwości

Co musi obsługiwać Twoja architektura? Cechy architektury (możliwości architektury) są podstawowymi elementami składowymi każdego systemu. Bez nich nie można podejmować decyzji architektonicznych, wybierać stylu architektury, a w wielu przypadkach nawet tworzyć logicznej architektury. W tym rozdziale dowiesz się, jak zdefiniować niektóre z bardziej powszechnych cech (takich jak skalowalność, niezawodność i testowalność), jak wpływają one na architekturę oprogramowania, jak pomagają w podejmowaniu decyzji architektonicznych i jak określić, które z nich są ważne w konkretnej sytuacji. Gotowy na dodanie pewnych możliwości do swojej architektury oprogramowania?



Cieszymy się razem!	40
Rozmowy w boksie	41
Czym są cechy architektury?	42
Definiowanie cech architektury	43
Cechy są aspektami projektu niezwiązanymi z dziedziną	44
Cechy mają wpływ na strukturę architektury	45
Ograniczaj cechy, by zapobiec nadmiernemu projektowaniu	46
Rozważ możliwości jawne i niejawne	48
Międzynarodowe Zoo Możliwości	49
Cechy architektury dotyczące procesu	50
Strukturalne cechy architektury	51
Operacyjne cechy architektury	52
Przekrojowe cechy architektury	53
Wyodrębnianie cech architektury z dziedziny problemu	58
Wyodrębnianie cech architektury ze świadomości środowiskowej	59
Określanie cech architektury na podstawie holistycznej wiedzy o dziedzinie	59
Złożone cechy architektury	61
Priorytety mają swój kontekst	62
Zagubiony w przekładzie	64
Cechy architektury i komponenty logiczne	66
Równoważenie kwestii związanych z dziedziną i cech architektury	67
Ograniczanie cech architektury	68

3

Dwa prawa architektury oprogramowania

Kompromisy... nic, tylko kompromisy!

Co się dzieje, gdy nie istnieje coś, co można by uznać za „najlepsze praktyki”?

Tym, co w najlepszych praktykach jest najlepsze, jest to, że są one sposobami osiągania określonych celów, które w znacznym stopniu nie są obciążone żadnym ryzykiem. Są one określane „najlepszymi” (a nie „lepszymi” lub „dobrymi”) nie bez powodu — wiadomo, że działają, więc dlaczego by ich nie użyć? Ale jedną rzeczą, której szybko nauczysz się o architekturze oprogramowania, jest to, że w jej przypadku nie ma czegoś takiego jak najlepsze praktyki. Będziesz musiał dokładnie przeanalizować każdą sytuację, aby podjąć decyzję, a w odniesieniu do tych decyzji komunikować nie tylko „co”, ale także „dlaczego”.

Jak więc poruszać się po tym nowym obszarze? Na szczęście masz do dyspozycji prawa architektury oprogramowania. Ten rozdział pokazuje, jak podczas podejmowania decyzji analizować kompromisy. Pokażemy również, jak tworzyć dokument ADR — zapisy decyzji architektonicznych — i w nim utrwaląć „jak” oraz „dlaczego” decyzji. Pod koniec tego rozdziału będziesz dysponował narzędziami pozwalającymi poruszać się po niepewnym terytorium, jakim jest architektura oprogramowania.



Wszystko zaczyna się od aplikacji ze sneakersami	78
Co już wiemy?	80
Komunikacja z usługami serwerowymi	82
Analiza kompromisów	83
Analiza kompromisów: wersja z kolejkami	84
Analiza kompromisów: wersja z tematami	85
Pierwsze prawo architektury oprogramowania	86
Zawsze wszystko sprowadza się do kompromisów	88
Podejmowanie decyzji architektonicznych	89
Co jeszcze sprawia, że decyzja ma charakter architektoniczny?	90
Drugie prawo architektury oprogramowania	92
Dokumenty ADR — zapisy decyzji architektonicznych	93
Pisanie ADR: wybór odpowiedniego tytułu	95
Pisanie dokumentów ADR: jaki masz status?	96
Pisanie dokumentów ADR: określanie kontekstu	100
Pisanie dokumentów ADR: przedstawienie decyzji	101
Pisanie dokumentów ADR: rozważenie konsekwencji	103
Pisanie dokumentów ADR: zapewnianie nadzoru	105
Pisanie dokumentów ADR: uwagi końcowe	105
Korzyści ze stosowania dokumentów ADR	108
Sneakersowy Sezam okazał się sukcesem	109

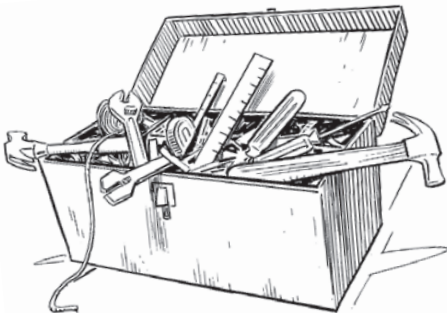
4

Komponenty logiczne
Elementy konstrukcyjne

Gotowy, by rozpocząć tworzenie architektury? Nie jest to tak łatwe, jak by się mogło wydawać, a jeśli nie zrobisz tego poprawnie, Twój system może się „rozsypać” — podobnie jak źle zaprojektowany wieżowiec lub most.

W tym rozdziale pokażemy kilka różnych rozwiązań problemu identyfikowania i tworzenia *komponentów logicznych* — funkcjonalnych elementów składowych systemów oprogramowania, opisujących sposób, w jaki poszczególne fragmenty systemu pasują do siebie nawzajem. Stosowanie opisanych tu technik ułatwi Ci stworzenie solidnej architektury — fundamentu, na którym będziesz w stanie zbudować udany system oprogramowania.

A zatem... załóż kask i rękawice, przygotuj narzędzia i zaczynajmy.



Ponowna prezentacja komponentów logicznych	118
Nazwij ten komponent	119
Aukcje na przygody online	120
Architektura logiczna a fizyczna	121
Tworzenie architektury logicznej	123
Krok 1. Określenie początkowych podstawowych komponentów	124
Podejście bazujące na przepływie pracy	126
Podejście bazujące na aktorach i akcjach	128
Pułapka encji	130
Krok 2. Przypisanie wymagań do komponentów	132
Krok 3. Analiza ról i odpowiedzialności	134
Dbanie o spójność	135
Krok 4. Analiza cech	136
Komponent Rejestracja oferty	138
Powiązania komponentów	139
Powiązania doprowadzające	140
Powiązania odprowadzające	141
Pomiar powiązań	142
System o ścisłych powiązaniach	144
Stosowanie prawa Demeter	145
Akt równoważenia	147
Kilka ostatnich słów o komponentach	148
Nazwij ten komponent	149

5

Style architektury

Kategoryzacja i filozofie

Istnieje wiele różnych stylów architektury. Każdy z nich powstał nie bez powodu i ma swoją własną filozofię dotyczącą tego, jak i kiedy powinien być używany. Zrozumienie filozofii danego stylu pomoże Ci ocenić, czy jest on odpowiedni dla danej dziedziny. Informacje zamieszczone w tym rozdziale stanowią fundament solidnej wiedzy o różnych rodzajach stylów architektury oprogramowania (które opiszemy znacznie dokładniej w dalszej części niniejszej książki). Jego lektura pomoże Ci zrozumieć wszelkie style architektury oprogramowania, z którymi będziesz miał okazję się zetknąć.

Dołożymy zatem ten ostatni element układanki, co Ty na to?

Istnieje wiele stylów architektury	160
Świat stylów architektury	161
Podział: techniczny kontra dziedzinowy	162
Model wdrożenia: monolityczny kontra rozproszony	164
Monolityczne modele wdrażania: zalety	166
Monolityczne modele wdrażania: wady	167
Rozproszone modele wdrażania: zalety	168
Rozproszone modele wdrażania: wady	169
A oto i podsumowanie!	172



6

Architektura warstwowa

Separując zagadnienia

Co zrobić, jeśli problem jest prosty, a kluczowe znaczenie ma czas?

Czy w ogóle warto zwracać sobie głowę architekturą? To zależy od tego, jak długo to, co aktualnie budujesz, ma działać. Jeśli jest to produkt jednorazowego użytku, w ogóle się nie przejmuj. Jeśli natomiast ma być używany przez dłuższy czas, to wybierz możliwie jak najprostszą architekturę, która zapewni jakąś organizację i korzyści, lecz nie będzie wywierać dużego wpływu na szybkość dostarczania produktu na rynek. Takie możliwości zapewnia *architektura warstwowa* — jest ona łatwa do zrozumienia oraz wdrożenia i pozwala na korzystanie z wzorców projektowych, które programiści już znają. Przyjrzyjmy się jej zatem warstwa po warstwie.

Naan Mniaam!: zbieranie wymagań	176
Wzorce projektowe — przypomnienie	178
Warstwy wzorca MVC	179
Warstwa po warstwie	182
Przekształcanie warstw na kod	183
Dziedziny, komponenty i warstwy	185
Zalety architektury warstwowej	188
Warstwy w realu: architektury fizyczne	189
Kompromisy architektury fizycznej	190
Ostatnie ostrzeżenie dotyczące zmian dziedziny	193
Supermoce architektury warstwowej	194
Kryptonit architektury warstwowej	195
Architektura warstwowa w ocenach	196
Podsumowanie	198



7

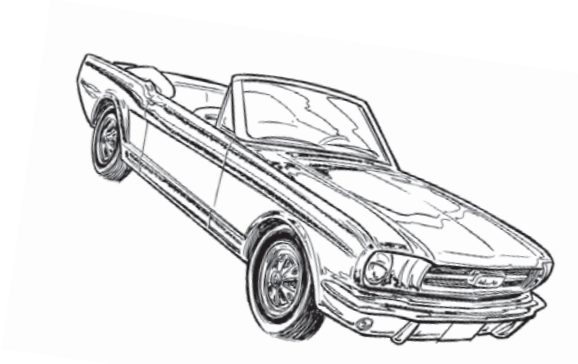
Modularne monolity

Bazujące na dziedzinnie

Monolity można budować na więcej niż jeden sposób. Do tej pory spotkałeś się z architekturą warstwową, która zwraca uwagę na aspekty *techniczne*. Twoja przygoda z architekturą warstwowego monolitu może być długa i owocna, ale kiedy zmiany zaczną dotyczyć komunikacji i koordynacji pomiędzy różnymi zespołami, może się okazać, że będziesz potrzebował nieco więcej mocy i możliwości — a może nawet zupełnie innego stylu architektury.

W tym rozdziale przyjrzymy się stylowi architektury nazywanemu *modularnym monolitem*. Charakteryzuje się on tym, że aplikacje są dzielone w oparciu o *kwestie biznesowe*, a nie techniczne. Podczas lektury tego rozdziału dowiesz się, co oznacza taki sposób podziału, na co zwrócić uwagę w przypadku stosowania tego stylu architektury oraz jakie kompromisy się z nim wiążą. A zatem wypróbujmy ten modularny monolit, co Ty na to?

Modularny monolit?	207
Dziedziczne problemy zmiany	209
Dlaczego modularne monolity?	210
Pokażcie mi kod!	212
Dbanie o modularność modułów	215
Rozszerzanie modularyzacji na bazy danych	219
Zwracaj uwagę na złączenia	221
Supermoce modularnych monolitów	222
Kryptonit modularnych monolitów	223
Modularny monolit w ocenach	224
Naan Mniaam! dostarcza pizze!	226



8

Architektura mikrojądra

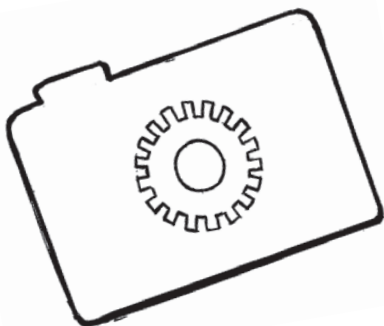
Konstruowanie dostosowań

Możesz tworzyć niestandardowe doświadczenia, możliwość po możliwości.

Niektóre style architektury są szczególnie dobrze dostosowane do niektórych możliwości, a jeśli chodzi o dostosowywanie, to prawdziwym mistrzem świata jest architektura mikrojądra. Oprócz tego świetnie się ona sprawdza w bardzo wielu różnych aplikacjach. Gdy zrozumiesz ten styl architektury, zaczniesz go widzieć wszędzie!

Przyjrzyjmy się zatem architekturze, która pozwala użytkownikom działać *po swojemu*.

Korzyści zapewniane przez RecyklKings	234
Dwie części architektury mikrojądra	237
Spektrum „mikrojądrowości”	239
Jądro usługi oceny urządzeń	241
Wtyczki hermetyzowane kontra wtyczki rozproszone	243
Komunikacja z wtyczkami	245
Kontrakty wtyczek	250
RecyklKings stawiają na ekologię	251
Supermoce mikrojądra	252
Kryptonit mikrojądra	253
Mikrojądro w ocenach	254
Podsumowanie	256



9

Zrób to sam

Aplikacja podróżnicza LuzTravel

Gotowy na rozszerzenie swojej podróży ku poznaniu architektury oprogramowania?

W tym rozdziale wcielisz się w rolę architekta oprogramowania. Będziesz określać cechy architektury, budować architekturę logiczną, podejmować decyzje architektoniczne i decydować, czy użyć architektury warstwowej, modułowej, czy mikrojądra. Ćwiczenia zamieszczone w tym rozdziale dadzą Ci kompleksowy obraz tego, co robi architekt oprogramowania, i pokażą, jak wiele już się nauczyłeś. Przygotuj się do stworzenia architektury dla start-upu budującego witrynę przeznaczoną do integracji i organizacji podróży. *Bon voyage* — życzymy udanej podróży podczas tworzenia architektury.

Ułatwianie podróżowania	262
Przepływ pracy użytkownika LuzTravel	263
Planowanie architektury	264
Przewodnik architekta	265
Krok 1. Określenie cech architektury	266
Krok 2. Identyfikacja komponentów logicznych	268
Krok 3. Wybór stylu architektury	270
Krok 4. Dokumentowanie podjętych decyzji	272
Krok 5. Rysowanie diagramu architektury	274
Nie ma dobrych (ani złych) odpowiedzi	276

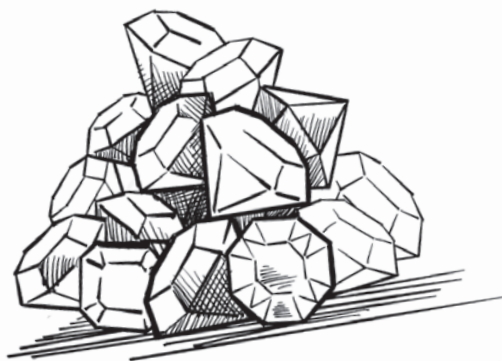


10

Architektura mikrousług

Kawałek po kawałku

Jak ułatwić sobie zmienianie architektury? Obecnie biznes zmienia się szybciej niż kiedykolwiek, a architektury oprogramowania muszą za nim nadążyć. W tym rozdziale dowiesz się, jak stworzyć elastyczną architekturę, która może być modyfikowana wraz ze zmianami biznesowymi, zapewni możliwość skalowania systemu wraz z rozwojem firmy i pozwoli mu działać nawet w przypadku awarii jego części. Zaintrygowany? Mamy nadzieję, że tak, ponieważ w tym rozdziale pokażemy Ci *mikrousługi* — styl architektury, który zapewnia wszystkie te możliwości i parę innych. Naszą podróż przez krainę mikrousług odbędziemy... cóż... kawałek po kawałku.

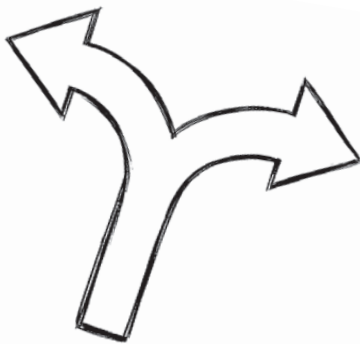


Jak się dzisiaj czujemy?	288
Czym jest mikrousługa?	291
To są moje dane, nie Twoje	292
Jak małe jest „mikro”?	294
Siły dezintegrujące	296
Dlaczego mielibyśmy zmniejszać mikrousługi?	297
Siły integrujące	298
Dlaczego mielibyśmy powiększać mikrousługi?	299
Wszystko sprowadza się do równowagi	300
Współdzielenie funkcjonalności	303
Wielokrotne użycie kodu dzięki zastosowaniu współdzielonej usługi	304
Wielokrotne użycie kodu dzięki zastosowaniu współdzielonej biblioteki	305
Zarządzanie przepływami pracy	309
Orkiestracja: dyrygowanie mikrousługami	310
Choreografia: zatańczmy	312
Supermoce architektury mikrousług	316
Kryptonit architektury mikrousług	317
Mikrousługi w ocenach	318
Podsumowanie	320

11

Architektura sterowana zdarzeniami
Asynchroniczne przygody

Co by było, gdyby Twoja architektura mogła robić wiele rzeczy w tym samym czasie? W miarę jak firmy rozwijają się i odnoszą coraz większe sukcesy, muszą także zapewniać możliwość obsługi coraz większej liczby użytkowników, i to bez spowalniania lub zawieszania się systemów. W tym rozdziale dowiesz się, jak projektować wydajne systemy, które można skalować wraz z rozwojem firmy. Przygotuj się na *architekturę sterowaną zdarzeniami*, bardzo popularny rozproszony styl architektury. Jest ona bardzo szybka, wysoce skalowalna i łatwa do rozbudowy, choć z drugiej strony jest także dość złożona. W tym rozdziale poznasz wiele nowych pojęć, w tym takie rzeczy jak zdarzenia, komunikaty i komunikacja asynchroniczna — ich znajomość jest niezbędna, jeśli chcesz nauczyć się tworzenia architektury, która może robić wiele rzeczy naraz. A więc zapnij pasy: wyruszmy na asynchroniczną przygodę z architekturą sterowaną zdarzeniami.



Zbyt wolno	332
Przyspieszanie obsługi	333
Der Nile rozwija się szybciej niż kiedykolwiek	334
Czym jest zdarzenie?	336
Zdarzenia a komunikaty	338
Zdarzenia inicjujące i pochodne	340
Czy ktokolwiek mnie słucha?	342
Komunikacja asynchroniczna	343
Odpal i zapomnij	345
Asynchroniczna rządzi	347
Synchroniczna rządzi	349
Topologie baz danych	351
Monolityczna baza danych	352
Bazy danych podzielone w oparciu o dziedzinę	354
Po bazie na usługę	356
Architektura EDA kontra mikrousługi	360
Hybrydy: mikrousługi sterowane zdarzeniami	364
Supermoce architektury sterowanej zdarzeniami	366
Kryptonit architektury sterowanej zdarzeniami	367
Architektura sterowana zdarzeniami w ocenach	368
Scalenie tego wszystkiego w całość	370
Podsumowanie	371

12

Zrób to sam

Sprawdzanie swojej wiedzy

Czy jesteś gotowy, by przetestować swoją wiedzę i umiejętności w zakresie tworzenia architektury rozproszonej? W tym rozdziale ponownie wcielisz się w rolę architekta oprogramowania. Będziesz określać cechy architektury, budować architekturę logiczną, podejmować decyzje architektoniczne i decydować, czy użyć mikrousług, czy architektury sterowanej zdarzeniami. Ćwiczenia zamieszczone w tym rozdziale dadzą Ci kompleksowy obraz tego, co robi architekt oprogramowania, i pokażą, jak wiele się nauczyłeś. Przygotuj się do stworzenia architektury dla systemu przeprowadzania standaryzowanych testów dla uczniów o nazwie Testorama. Powodzenia — mamy nadzieję, że Twoja architektura zostanie oceniona na szóstkę!

Witamy w Testoramie	384
Przebieg realizacji testów	385
Planowanie architektury	386
Przewodnik architekta	387
Krok 1. Określenie cech architektury	388
Krok 2. Identyfikacja komponentów logicznych	390
Krok 3. Wybór stylu architektury	392
Krok 4. Dokumentowanie podjętych decyzji	394
Krok 5. Narysowanie diagramu architektury	396
Nie ma dobrych (ani złych) odpowiedzi	398



Dodatek: pozostałości

**Sześć najważniejszych zagadnień,
których nie opisaliśmy**

O architekturze oprogramowania można powiedzieć znacznie więcej.

Obiecujemy, że to już jest prawie sam koniec tej książki. Jednak jej lektura to dopiero pierwszy krok na drodze do myślenia architektonicznego, a nie czulibyśmy się w porządku, pozwalając Ci odejść bez dodatkowego przygotowania. Dlatego w tym dodatku zebraliśmy i przedstawiliśmy kilka dodatkowych smakowitych kąsków. Każde z opisanych tu zagadnień zasługuje na tyle samo uwagi co inne tematy przedstawione wcześniej w książce. Jednak tym razem chcemy jedynie dość ogólnie pokazać Ci, o co w nich chodzi. I ostatnia sprawa: to faktycznie *jest* koniec tej książki. Oczywiście pomijając indeks — w nim też jest parę dobrych stron!

Nr 1. Architekt piszący kod	408
Nr 2. Oczekiwanie względem architektów	410
Nr 3. Miękkie umiejętności architektów	412
Nr 4. Techniki tworzenia diagramów	414
Nr 5. Głębokość a szerokość wiedzy	416
Nr 6. Praktykowanie architektury w formie kata	418

Skorowidz	421
-----------	-----



1. Architektura oprogramowania bez tajemnic

Zaczynamy!



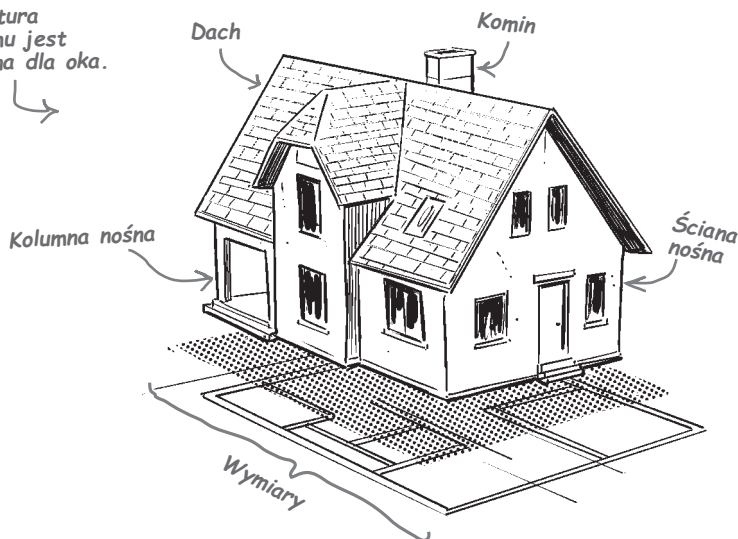
Architektura oprogramowania ma fundamentalne znaczenie dla sukcesu systemu. Ten rozdział jest poświęcony omówieniu architektury oprogramowania. Czytając go, zrozumiesz, czym są wymiary architektury, i poznasz różnice pomiędzy architekturą oprogramowania a projektowaniem. Dlaczego jest to ważne? Ponieważ zrozumienie i stosowanie praktyk architektonicznych pomaga budować bardziej efektywne i poprawne oprogramowanie — oprogramowanie, które nie tylko lepiej funkcjonuje, ale także spełnia potrzeby i łagodzi obawy biznesu, a co więcej, zapewnia ciągłe poprawne działanie mimo bezustannych zmian, którym podlegają środowiska biznesowe i techniczne. Tak więc, bez dalszej zwłoki, zaczynamy.

Budowanie zrozumienia architektury oprogramowania

Aby lepiej zrozumieć architekturę oprogramowania, wyobraź sobie typowy dom w Twojej okolicy. Struktura takiego domu to jego *architektura* — określają ją takie aspekty jak: kształt, liczba pokoi i pięter, wymiary i tak dalej. Dom jest zwykle reprezentowany przez plan budynku, który zawiera wszystkie linie i pola niezbędne do określenia, jak dany dom zbudować. Elementy konstrukcyjne, takie jak te pokazane poniżej, są trudne i kosztowne do późniejszej zmiany i są ważnymi elementami domu.

Metafora budynku jest często wykorzystywana podczas wyjaśniania architektury oprogramowania.

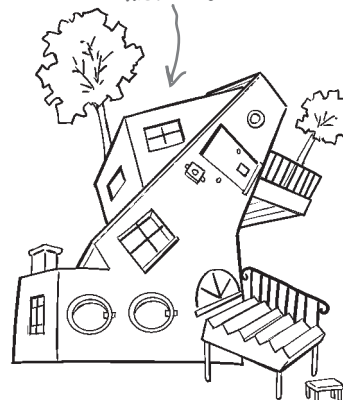
Architektura tego domu jest przyjemna dla oka.



Architektura jest kluczowa dla budowania domu. Czy można sobie wyobrazić budowę domu bez architektury? Mogłoby się okazać, że będzie on wyglądał jak dom przedstawiony obok.

Architektura jest również niezbędna do tworzenia systemów oprogramowania. Czy kiedykolwiek natknąłeś się na system, który nie zapewnia dobrej skalowalności, jest zawodny lub trudny w utrzymaniu? Jeśli tak, to prawdopodobnie na etapie tworzenia nie położono wystarczającego nacisku na przygotowanie jego architektury.

Ten dom nie tylko jest paskudny, lecz także niefunkcjonalny.



Ćwiczenie

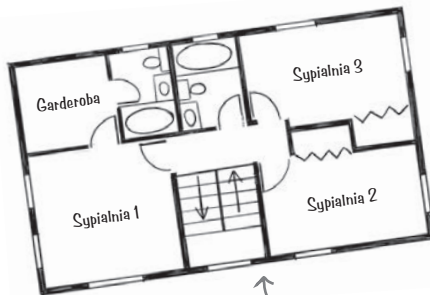
Kolejną przydatną i popularną metaforą architektury oprogramowania jest ogrodnictwo. W pustym obszarze w dolnej części tej ramki spróbuj opisać, w jaki sposób planowanie ogrodu może się odnosić do architektury oprogramowania. Na końcu rozdziału możesz sprawdzić, jakie są nasze spostrzeżenia na ten temat.

→ Rozwiązanie znajdziesz na stronie 28

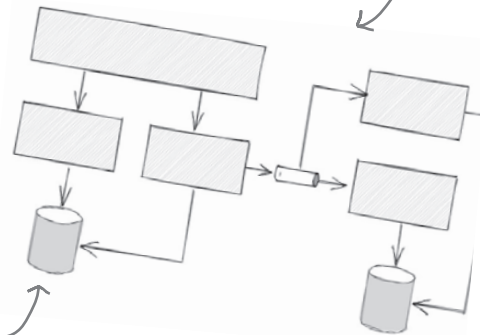
Plan budynku i architektura oprogramowania

Być może zastanawiasz się, w jaki sposób plany Twojego domu odnoszą się do architektury oprogramowania. Są one reprezentacją budowanej rzeczy. Jak więc wygląda „plan budynku” w kontekście systemu oprogramowania? To oczywiście linie i prostokąty.

Plan budynku określa strukturę domu — tworzące go pokoje, ściany, schody i tak dalej — w ten sam sposób, w jaki diagram architektury oprogramowania określa strukturę programu (jego interfejsy użytkownika, wykorzystywane usługi, bazy danych i protokoły komunikacyjne). Oba te plany zawierają wytyczne i ograniczenia, a także wizję końcowego rezultatu.



Oba schematy przedstawiają plany konstrukcyjne.



Ciekawostka: plan budynku był kiedyś nazywany „światłokopią” (ang. blueprint), ale termin ten jest obecnie przestarzały (przynajmniej w branży architektoniczno-budowlanej).



Jakie cechy domu, *strukturalne* i związane z jego *architekturą*, jesteś w stanie wymienić? Nasze przemyślenia znajdziesz na końcu tego rozdziału.

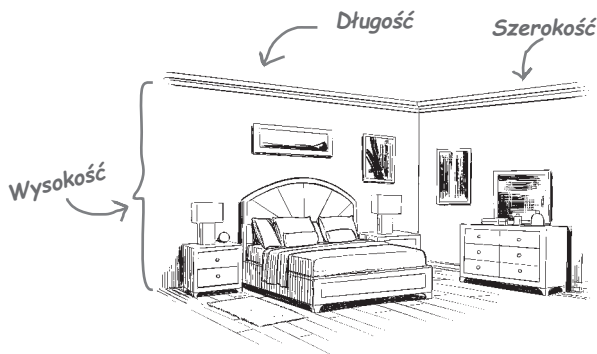
Tutaj zapisz swoją odpowiedź.

→ Rozwiązanie znajdziesz na stronie 28

Czy zauważyłeś, że plan piętra powyższego domu nie określa szczegółów pomieszczeń — takich jak rodzaj podłogi (dywan lub drewno z drzewa liściastego), kolor ścian i miejsce, w którym w sypialni należy ustawić łóżko? To dlatego, że te zagadnienia nie są *strukturalne*. Innymi słowy, nie określają one *architektury* domu, ale raczej jego *projekt*.

Nie martw się — w dalszej części tego rozdziału dowiesz się znacznie więcej o tym rozróżnieniu. W tej chwili skup się na samej strukturze projektowanego „czegoś” — innymi słowy, na jego architekturze.

Wymiary architektury oprogramowania



Większość rzeczy wokół nas jest wielowymiarowa. Możesz na przykład opisać konkretny pokój w swoim domu, mówiąc, że ma on 5 metrów długości i 4 metry szerokości, a sufit znajduje się na wysokości 2,5 metra. Zauważ, że aby poprawnie opisać ten pokój, musisz określić wszystkie trzy wymiary: wysokość, długość i szerokość.

Również architekturę oprogramowania można opisywać za pomocą jej wymiarów. Różnica polega na tym, że architektura oprogramowania ma **cztery wymiary**.

1 Cechy architektury

Ten wymiar opisuje, jakie aspekty systemu architektura musi wspierać — chodzi o takie zagadnienia jak: skalowalność, łatwość testowania, dostępność techniczna (ang. *availability*) itp.

2 Decyzje architektoniczne

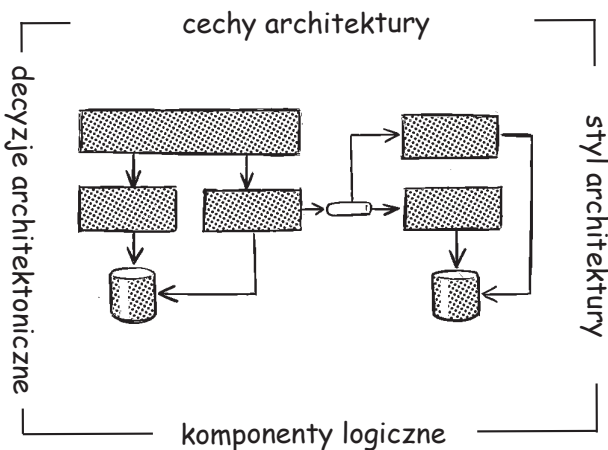
Ten wymiar obejmuje ważne decyzje, które mają długoterminowe lub znaczące implikacje dla systemu — na przykład rodzaj używanej bazy danych, liczbę usług i sposób, w jaki się one ze sobą komunikują.

3 Komponenty logiczne

Ten wymiar opisuje elementy składowe funkcjonalności systemu i ich wzajemne interakcje. Na przykład system handlu elektronicznego może mieć komponenty do zarządzania magazynem, przetwarzania płatności itd.

4 Styl architektury

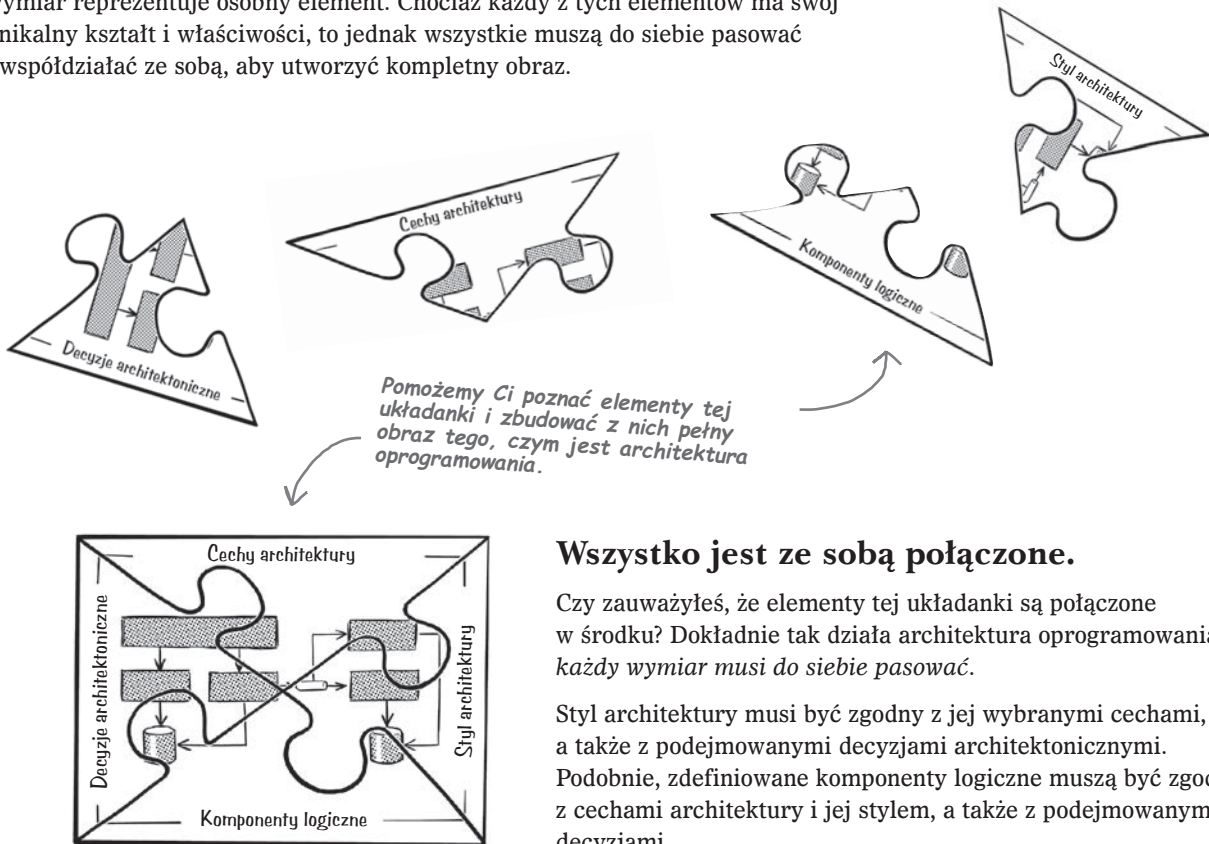
Ten wymiar definiuje ogólny fizyczny kształt i strukturę systemu oprogramowania w taki sam sposób, w jaki plan budynku definiuje ogólny kształt i strukturę domu.



W dalszej części tej książki poznasz pięć najpopularniejszych stylów architektury oprogramowania.

Rozgrzanie wymiarów

O architekturze oprogramowania można myśleć jak o układance, w której każdy wymiar reprezentuje osobny element. Chociaż każdy z tych elementów ma swój unikalny kształt i właściwości, to jednak wszystkie muszą do siebie pasować i współdziałać ze sobą, aby utworzyć kompletny obraz.



Wszystko jest ze sobą połączone.

Czy zauważyłeś, że elementy tej układanki są połączone w środku? Dokładnie tak działa architektura oprogramowania: każdy wymiar musi do siebie pasować.

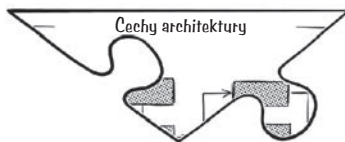
Styl architektury musi być zgodny z jej wybranymi cechami, a także z podejmowanymi decyzjami architektonicznymi. Podobnie, zdefiniowane komponenty logiczne muszą być zgodne z cechami architektury i jej stylem, a także z podejmowanymi decyzjami.

Nie istnieją
głupie pytania

P: Czy podczas tworzenia architektury potrzebne są wszystkie cztery wymiary, czy też można pominąć niektóre z nich, jeśli nie ma na to czasu?

O: Niestety, nie można pominąć żadnego z tych wymiarów — wszystkie są wymagane do stworzenia i opisu architektury. Jednym z często popełnianych przez architektów oprogramowania błędów jest używanie podczas opisywania swojej architektury tylko jednego lub dwóch z tych wymiarów. Stwierdzenie takie jak „nasza architektura to mikrouslugi” opisuje jeden wymiar architektury — konkretnie jej styl — ale pozostawia zbyt wiele pytań bez odpowiedzi. Na przykład jakie cechy architektury są krytyczne dla sukcesu systemu? Jakie są jego logiczne komponenty (funkcjonalne bloki konstrukcyjne)? Jakie główne decyzje zostały podjęte w odniesieniu do sposobu wdrożenia architektury?

Pierwszy wymiar: cechy architektury



Cechy architektury stanowią podstawę architektury systemów oprogramowania. Bez nich nie można podejmować decyzji architektonicznych ani analizować ważnych kompromisów.

Wyobraź sobie, że próbujesz wybrać jeden z dwóch domów. Jeden z nich jest przestronny, ale znajduje się obok ruchliwej, hałaśliwej autostrady. Drugi dom znajduje się w ładnej, cichej okolicy, ale jest znacznie mniejszy. **Która cecha**

jest dla Ciebie ważniejsza — wielkość domu czy poziom hałasu i natężenia ruchu w okolicy? Bez określenia tych priorytetów nie będziesz w stanie dokonać właściwego wyboru.

To samo dotyczy architektury oprogramowania. Załóżmy, że musisz zdecydować, jakiego rodzaju bazy danych użyć w swoim nowym systemie. Czy powinna to być relacyjna baza danych, prosta baza danych przechowująca klucze i wartości, czy też złożona, grafowa baza danych? Odpowiedź będzie zależała od tego, jakie cechy architektury są dla Ciebie krytyczne. Na przykład jeśli potrzebujesz możliwości szybkiego wyszukiwania (tę cechę nazywamy **wydajnością**), to możesz wybrać grafową bazę danych; z kolei tradycyjna relacyjna baza danych może być lepsza, jeśli chcesz zachować powiązania pomiędzy danymi (nazywamy to **integralnością danych**).

wydajność
Czas potrzebny systemowi na przetworzenie żądania biznesowego.

dostępność techniczna
Czas sprawności systemu; zwykle mierzony w „dziewiątkach” (zatem 99,9% to trzy „dziewiątki”).

skalowalność
Zdolność systemu do utrzymania stałego czasu odpowiedzi i poziomu błędów w miarę wzrostu liczby użytkowników lub żądań.

Oto niektóre z bardziej powszechnych cech architektonicznych. W rozdziale 2. dowiesz się o nich znacznie więcej.



Ćwiczenie

Zaznacz to, co Twoim zdaniem można uznać za cechy architektury — coś, co jest wspierane przez *strukturę* systemu oprogramowania.

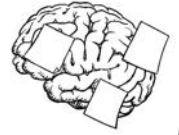
- ☐ Zmiana rozmiaru czcionki w oknie prezentującym interfejs użytkownika
- ☐ Szybkie wprowadzanie zmian
- ☐ Obsługa tysięcy jednocześnie pracujących użytkowników
- ☐ Szyfrowanie przechowywanych w bazie danych haseł użytkowników
- ☐ Interakcja z wieloma systemami zewnętrznymi w celu realizacji potrzeb biznesowych

→ Rozwiązanie znajdziesz na stronie 29

Termin *cechy architektury* może nie być Ci znany, ale nie oznacza to, że wcześniej o nim nie słyszałeś. Ogólnie rzecz biorąc, takie zagadnienia jak wydajność, skalowalność, niezawodność i dostępność techniczna są również znane jako wymagania niefunkcjonalne lub atrybuty jakości systemu. Jeśli o nas chodzi, to lubimy termin *cechy architektury*, ponieważ to faktycznie są cechy, które pozwalają zdefiniować charakter architektury i to, co musi ona wspierać.

Cechy architektury to możliwości, które mają krytyczne znaczenie lub są ważne dla powodzenia systemu.

**Zapamiętaj
to dobrze**



By soft projektować, zrozum cechy jego,
bez tego nie osiągniesz sukcesu pełnego!

Kto co robi ?

Oto szansa na sprawdzenie, ile już wiesz o wielu typowych cechach architektury. Czy potrafisz dopasować każdą z cech przedstawionych po lewej stronie do definicji podanej po prawej? Zauważysz, że definicji jest więcej niż cech, więc bądź ostrożny — nie wszystkie definicje mają pasującą nazwę.

Rozszerzalność

*To zrobiliśmy
za Ciebie.*

Zwinność

Współdziałanie

Odporność na błędy

Wykonalność

Uwzględnienie przy dokonywaniu wyborów architektonicznych ram czasowych, budżetów i umiejętności programistów.

Zdolność systemu do zachowania prawidłowego funkcjonowania innych jego części w przypadku wystąpienia błędów krytycznych.

Łatwość, z jaką system może zostać rozbudowany w celu obsługi dodatkowych możliwości i funkcjonalności.

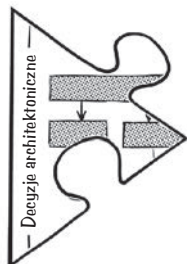
Czas potrzebny na uzyskanie odpowiedzi od użytkownika.

Zdolność systemu do szybkiego reagowania na zmiany (funkcja łatwości utrzymania, testowania i wdrażania).

Zdolność systemu do współdziałania i interakcji z innymi systemami w celu realizacji potrzeb biznesowych.

→ Odpowiedź znajdziesz na stronie 29

Drugi wymiar: decyzje architektoniczne



Decyzje architektoniczne to wybory dotyczące strukturalnych aspektów systemu, które mają długoterminowe lub znaczące konsekwencje. Ponieważ są to ograniczenia, będą uwzględniane przez zespół programistów podczas planowania i budowy systemu.

Czy nowy dom powinien mieć jedno, czy dwa piętra? Czy dach powinien być płaski, czy spadzisty? Czy powinienś zbudować duży, rozległy dom typu rancho? To są dobre przykłady decyzji architektonicznych, ponieważ dotyczą one *strukturalnego* aspektu domu.

Jak powinien wyglądać Twój dom? Ten rodzaj decyzji jest decyzją architektoniczną.



Możesz zdecydować, że interfejs użytkownika systemu nie powinien się komunikować bezpośrednio z bazą danych, a zamiast tego w celu pobierania i aktualizowania danych będzie musiał skorzystać z dodatkowych usług. Ta decyzja architektoniczna nakłada szczególne ograniczenia na rozwój interfejsu użytkownika, a także określa na potrzeby zespołu programistów, w jaki sposób inne komponenty powinny uzyskiwać dostęp do danych w bazie danych i je aktualizować.

Oto przykład pierwszej decyzji architektonicznej.

Decyzja architektoniczna

Interfejs użytkownika musi skorzystać z usługi dostępu do danych, aby odczytać lub zapisać informacje; nie może komunikować się bezpośrednio z bazą danych.

Ta decyzja architektoniczna narzuca ograniczenia i pełni rolę wskazówki.

Usługa dostępu do danych

Ten obrazek reprezentuje usługę. W tej książce zetkniesz się z nim wielokrotnie.



A to jest baza danych.

Znacznie więcej na temat decyzji architektonicznych dowiesz się w rozdziale 3.

Zapamiętaj to dobrze



Decyzje to dla programistów wskazówki o charakterze strukturalnym, często dotyczące zagadnień o znaczeniu kolosalnym.

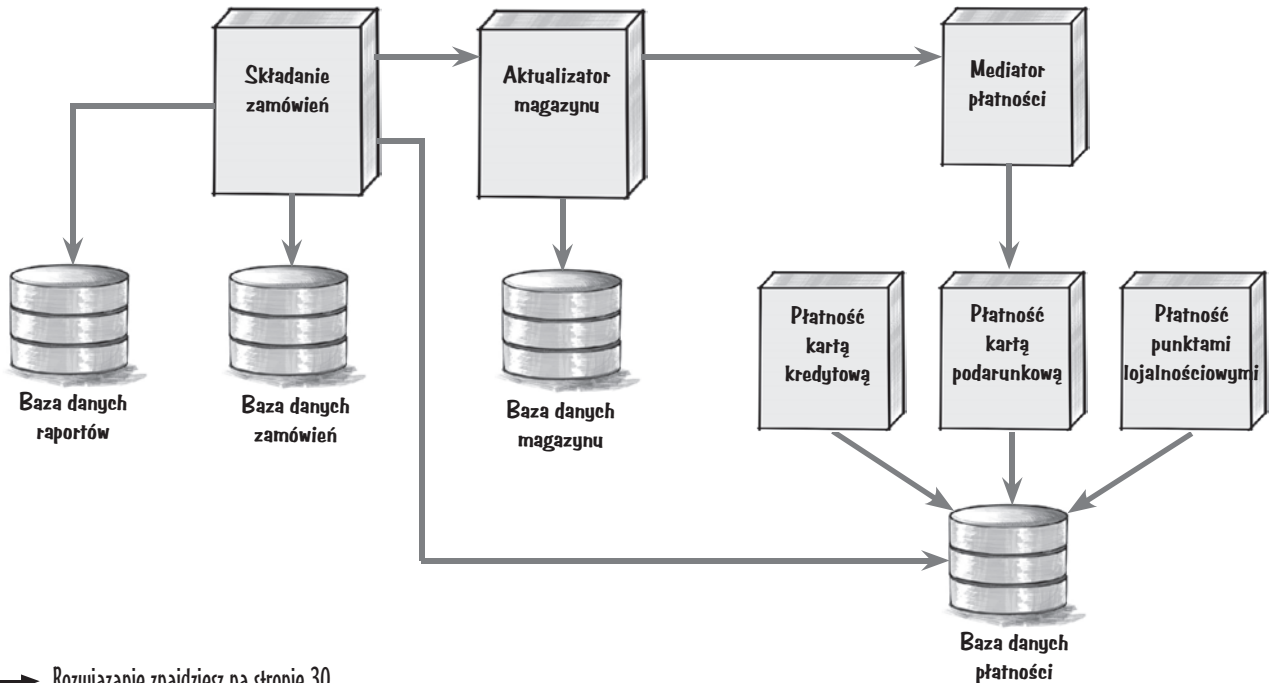
Nie jest rzadkością, że podczas tworzenia systemu oprogramowania zostanie podjętych i udokumentowanych kilkadziesiąt lub nawet więcej decyzji architektonicznych. Ogólnie rzecz biorąc, im większy i bardziej skomplikowany system, tym więcej będzie takich decyzji.

Bądź architektem



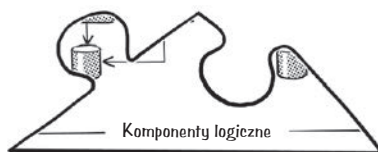
Twoim zadaniem jest wcielenie się w rolę architekta i zidentyfikowanie na poniższym diagramie jak najwięcej decyzji architektonicznych. Narysuj kółko wokół wszystkiego, co Twoim zdaniem może być decyzją architektoniczną, i napisz, jaka to może być decyzja.

Oto podpowiedź — czy masz pytania dotyczące tego, dlaczego pewne rzeczy są robione w taki, a nie inny sposób?



→ Rozwiązanie znajdziesz na stronie 30

Trzeci wymiar: komponenty logiczne

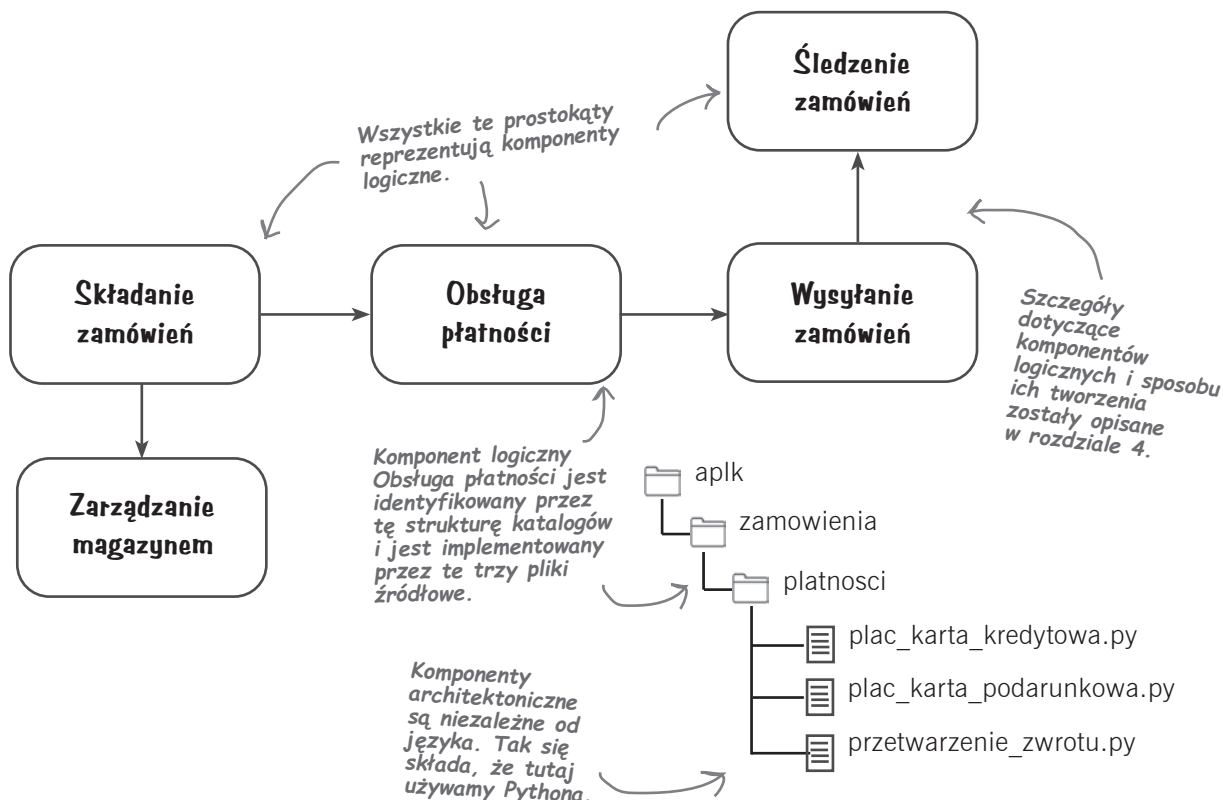
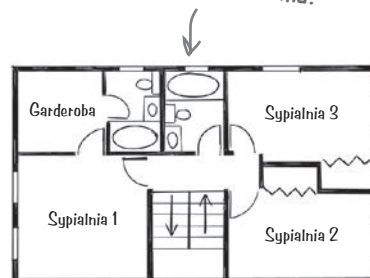


Komponenty logiczne są elementami składowymi systemu, podobnie jak pokoje są elementami składowymi domu. Komponent logiczny wykonuje pewną funkcję, taką jak przetwarzanie płatności za

zamówienie, zarządzanie magazynem produktów lub śledzenie zamówień.

Komponenty logiczne w systemie są zwykle reprezentowane przez katalog lub przestrzeń nazw. Na przykład katalog `aplk/zamowienia/platnosci` z odpowiadającą mu przestrzenią nazw `aplk.zamowienia.platnosci` identyfikuje komponent logiczny o nazwie Obsługa płatności. Kod źródłowy, który pozwala użytkownikom zapłacić za zamówienie, jest przechowywany w tym katalogu i używa tej przestrzeni nazw.

Te pomieszczenia stanowią podstawowe elementy składowe domu.



Zaostrz ołówek

Właśnie utworzyłeś następujące dwa komponenty dla nowego systemu, a Twój zespół programistów chce rozpocząć pisanie kodu klas, aby je zaimplementować. Czy możesz utworzyć dla nich strukturę katalogów, aby zespół mógł rozpocząć pracę nad kodem? Przejdź do końca rozdziału, aby zapoznać się z naszym rozwiązaniem.

**Profil
klienta**

**Preferencje
klienta**

*Swoje odpowiedzi
zapisz w pustym
obszarze w dolnej
części ramki.*

→ Rozwiązanie znajdziesz na stronie 31

Komponent logiczny powinien zawsze mieć dobrze zdefiniowaną rolę i odpowiedzialność w systemie — innymi słowy, jasną definicję tego, co robi.

Ten komponent jest odpowiedzialny za operację „znajdź i zapakuj”. Lokalizuje przedmioty w magazynie (to część „znajdź”), a następnie określa prawidłowy rozmiar pudełka dla zamówionych przedmiotów, aby można je było wysłać (to część „zapakuj”).

**Realizacja
zamówienia**

*Oto określenie roli
oraz odpowiedzialności
komponentu Realizacja
zamówienia*

**Zapamiętaj
to dobrze**



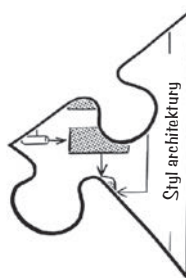
*Komponenty logiczne —
jak synergii pełne programów składowe,
gromadzące kod implementujący funkcje
biznesowe.*

Nie istnieją
głupie pytania

P: Jaka jest różnica pomiędzy funkcjonalnością systemu a dziedziną?

O: *Dziedzina* to problem, który starasz się rozwiązać, natomiast *funkcjonalność systemu* to sposób, w jaki próbujesz ten problem rozwiązać. Innymi słowy, dziedzina jest odpowiedzią na pytanie „co?”, a funkcjonalność systemu — odpowiedzią na pytanie „jak?”.

Czwarty wymiar: style architektury



Domy mają różne kształty, rozmiary i style. Chociaż istnieją domy wyglądające dziwacznie, większość z nich jest zgodna z określonym stylem, takim jak wiktoriański, ranczo lub Tudor. Styl domu wiele mówi o jego ogólnej strukturze. Na przykład domy typu ranczo mają zazwyczaj tylko jedną kondygnację; domy w stylu kolonialnym i Tudor mają zwykle kominy; a współczesne domy najczęściej mają płaskie dachy.

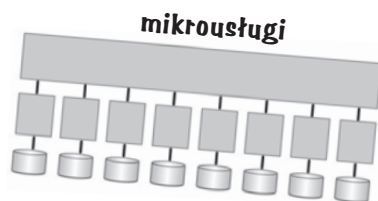
W jakim stylu został zbudowany dom, w którym mieszkasz?



Każdy region świata ma swój własny zestaw a domów — sprawdź je na stronie https://en.wikipedia.org/wiki/List_of_house_styles.

Style architektury definiują ogólny kształt i strukturę systemu oprogramowania, a każdy z nich ma swój własny, unikalny zestaw cech. Przykładowo styl architektury określany jako **mikrouslugi** zapewnia doskonałą skalowalność i wysoki poziom zwinności (ang. *agility*) — zdolności do szybkiego reagowania na zmiany, podczas gdy styl **warstwowy** jest mniej złożony i mniej kosztowny. Kolejny styl architektury — architektura **sterowana zdarzeniami** — zapewnia wysoki poziom skalowalności, jest bardzo szybki i responsywny.

Nie martw się — w dalszej części książki dowiesz się wszystkiego o tych stylach architektury. Każdemu z nich poświęciliśmy osobny rozdział.



mikrouslugi



architektura warstwowa



architektura sterowana zdarzeniami

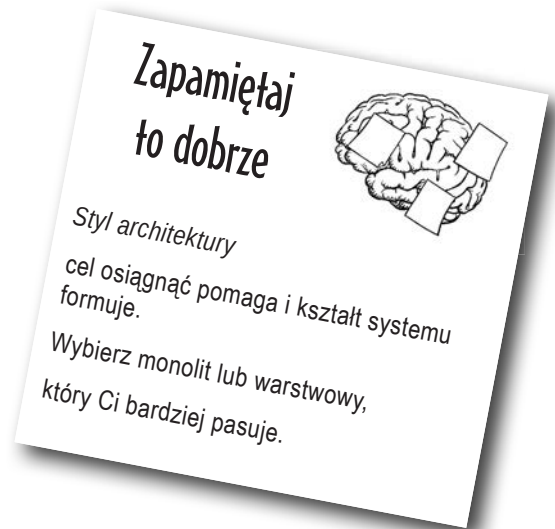
Istnieje wiele różnych stylów architektury oprogramowania, ale na szczęście nie tak wiele jak stylów budowania domów.

Ponieważ styl architektoniczny definiuje ogólny kształt i charakterystykę systemu, ważne jest, aby zrobić to dobrze za pierwszym razem. Dlaczego? Czy możesz sobie wyobrazić rozpoczęcie budowy jednopiętrowego domu na ranchu, a w połowie budowy zmienić zdanie i zdecydować, że zamiast tego zbudujesz trzypiętrowy wiktoriański dom? Byłoby to duże przedsięwzięcie, które prawdopodobnie przekroczyłoby budżet i wpłynęłoby na termin wprowadzenia się do domu.

Architektura oprogramowania nie jest inna. Nie jest łatwo zmienić monolityczną architekturę warstwową na mikrousługi. Podobnie jak w przypadku domu, byłoby to spore przedsięwzięcie.



Konwersja z rozległego domu typu rancho na wielopiętrowy dom wiktoriański byłaby niezwykle trudna, ponieważ mają one bardzo odmienne struktury.



W dalszej części książki pokażemy, jak prawidłowo wybrać styl architektury w oparciu o cechy, które są dla Ciebie ważne.

A to prowadzi nas z powrotem do wcześniejszego punktu — wszystkie wymiary architektury oprogramowania są ze sobą powiązane. Nie możesz wybrać stylu architektury, nie wiedząc, co ma dla nas kluczowe znaczenie.



MOC UMYŚŁU



Ciasno zwinięte ścięgna i mięśnie w nogach lwa umożliwiają mu osiągnięcie prędkości nawet do 80 kilometrów na godzinę i wykonywanie skoków na odległość 11 metrów. Ta cecha pozwala lwom przetrwać, gdyż dzięki niej są one w stanie z powodzeniem polować na szybkie zwierzęta.

Rozejrzyj się wokół siebie — co jeszcze ma strukturę lub kształt, które określają jego cechy i możliwości?

Ciekawostka: lew nie ma zbyt dużej wytrzymałości i może biegać szybko tylko na krótkich odcinkach. Jeśli będziesz w stanie biec dłużej niż goniący Cię lew, możesz przeżyć.

Kto co robi ?

Próbowaliśmy opisać naszą architekturę, ale wszystkie elementy układanki się pomieszały. Czy możesz nam pomóc dowiedzieć się, który wymiar za co odpowiada, dopasowując stwierdzenia po lewej stronie do wymiarów architektury oprogramowania widocznych po prawej? Uważaj — niektóre stwierdzenia nie pasują, ponieważ nie są związane z architekturą oprogramowania.

Ten system musi być dostępny dla naszych zagranicznych klientów.

Klienci narzekają na kolor tła nowego interfejsu użytkownika.

Właściciel produktu nalega, abyśmy jak najszybciej udostępniali klientom nowe funkcje i poprawki błędów.

Nasz system wykorzystuje architekturę sterowaną zdarzeniami.

W tym systemie musimy obsłużyć jednocześnie do 300 tysięcy użytkowników.

Pojedyncza usługa płatności zostanie podzielona na osobne usługi, po jednej dla każdego akceptowanego przez nas typu płatności.

Zamierzamy zacząć oferować punkty premiowe jako nową opcję płatności za zamówienia.

Rozbijamy klasę składanieZamowien na trzy mniejsze klasy w trzech plikach.

Interfejs użytkownika nie może komunikować się bezpośrednio z bazą danych.

Tu chodzi o dostępność.

Na to pytanie odpowiedzieliśmy za Ciebie.

Cechy architektury

Komponenty logiczne

Styl architektury

Decyzje architektoniczne

→ Rozwiązanie znajdziesz na stronie 32

Jeśli jestem odpowiedzialna za projekt systemu oprogramowania, czy oznacza to, że jestem również odpowiedzialna za jego architekturę? Czy to nie to samo?



Nie, architektura i projektowanie to co innego.

Architektura jest mniej związana z wyglądem, a bardziej ze strukturą, podczas gdy projekt jest mniej związany ze strukturą, a bardziej z wyglądem.

Kolor ścian pokoju, rozmieszczenie mebli i rodzaj podłogi (dywan lub drewno) są aspektami projektu, podczas gdy fizyczny rozmiar pokoju oraz rozmieszczenie drzwi i okien są częścią architektury – innymi słowy, *struktury* pokoju.

Pomyśl o typowej aplikacji biznesowej. Architektura lub struktura dotyczy sposobu, w jaki strony internetowe komunikują się z usługami serwerowymi i bazami danych w celu pobierania i zapisywania danych; natomiast projekt dotyczy wyglądu poszczególnych stron: kolorów, rozmieszczenia pól, używanych wzorców projektowych i tak dalej. Ponownie sprowadza się to do rozróżnienia pomiędzy strukturą a wyglądem.

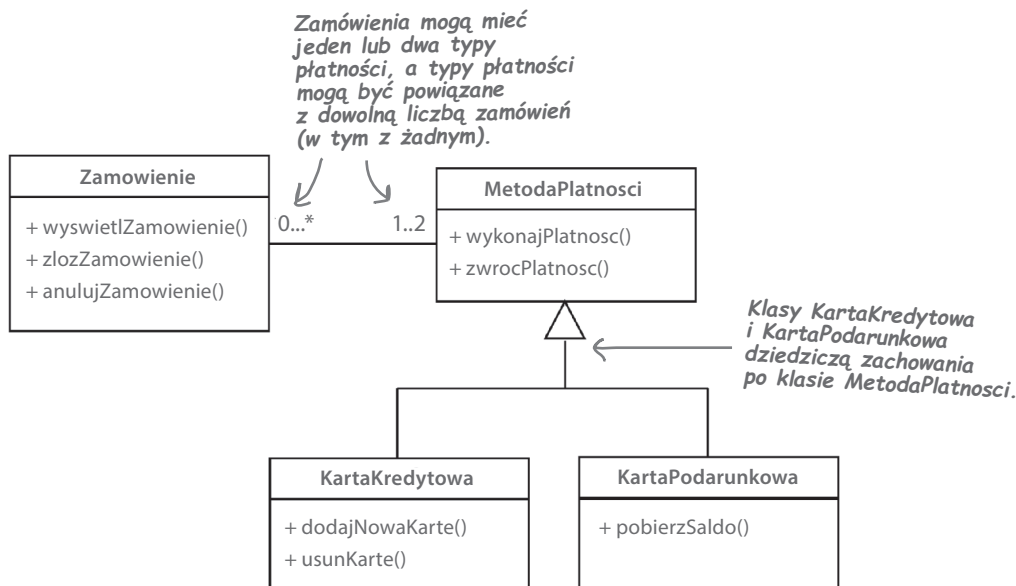
Twoje pytanie jest dobre, ponieważ określenie, co jest uważane za architekturę, a co za projekt, jest czasami mylące. Przeanalizujmy zatem te różnice.

Z punktu widzenia projektu

Założmy, że Twoja firma chce zastąpić swój przestarzały system przetwarzania zamówień nowym, zbudowanym na zamówienie, który lepiej odpowiada specyficznym potrzebom firmy. Klienci mogą składać zamówienia, a po ich złożeniu mogą je przeglądać lub anulować. Mogą zapłacić za zamówienie za pomocą karty kredytowej, karty podarunkowej lub obu tych metod płatności.



Z perspektywy projektu można przygotować diagram klas UML (ang. *Unified Modeling Language*), taki jak ten zamieszczony poniżej, aby pokazać, w jaki sposób poszczególne klasy współdziałają ze sobą, w celu zaimplementowania funkcjonalności obsługi płatności. Choć można napisać kod źródłowy, aby zaimplementować te pliki klas, ten projekt nie mówi nic o fizycznej strukturze kodu źródłowego — innymi słowy, o sposobie, w jaki te klasy zostaną zorganizowane i wdrożone.

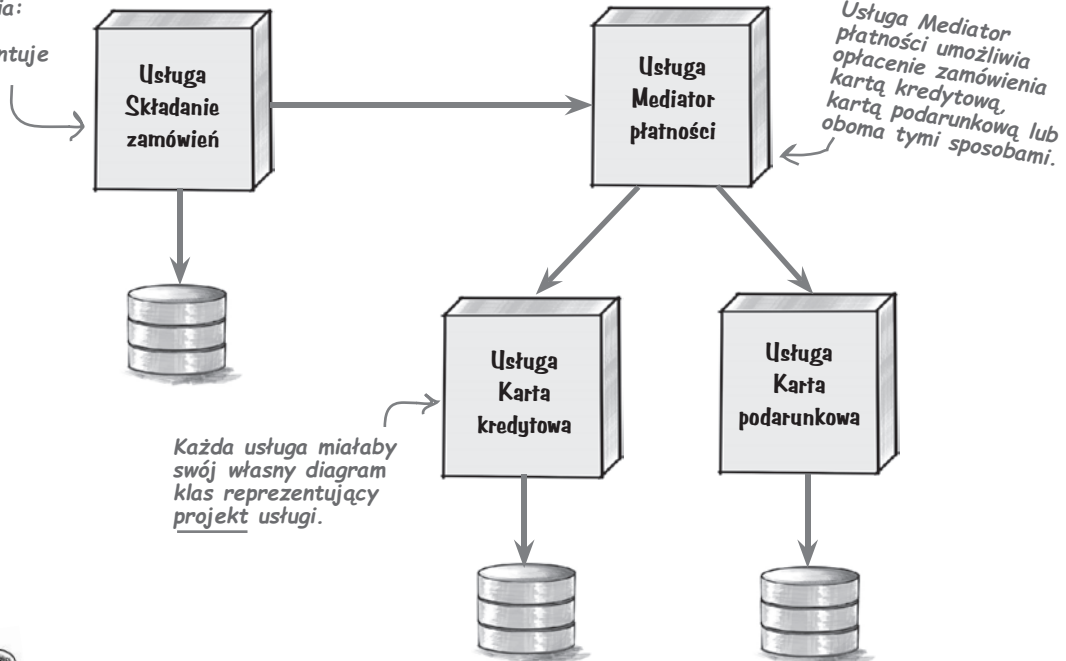


Z punktu widzenia architektury

W przeciwieństwie do projektowania, architektura dotyczy *struktury* systemu — takich rzeczy jak usługi, bazy danych i sposób, w jaki usługi komunikują się ze sobą oraz z interfejsem użytkownika.

Zastanówmy się jeszcze raz nad tym nowym systemem przetwarzania zamówień. Jak ten *system* powinien wyglądać? Z perspektywy *architektury* mógłbyś zdecydować się na utworzenie oddzielnych usług dla każdego typu płatności w ramach procesu płatności za zamówienie i zastosować usługę orkiestratora do zarządzania częścią systemu przetwarzającą płatności, tak jak pokazaliśmy na poniższym diagramie.

Dla przypomnienia:
każde z tych
pudełek reprezentuje
„usługę”.



Ćwiczenie

Zaznacz wszystko, co powinno być zawarte w diagramie narysowanym z *punktu widzenia architektury*.

- ☐ Jak usługi komunikują się ze sobą
- ☐ Platforma i język, w którym zaimplementowane są usługi
- ☐ Które usługi mają dostęp do których baz danych
- ☐ Ile jest usług i baz danych

→ Rozwiązanie znajdziesz na stronie 33

Spektrum pomiędzy architekturą a projektem

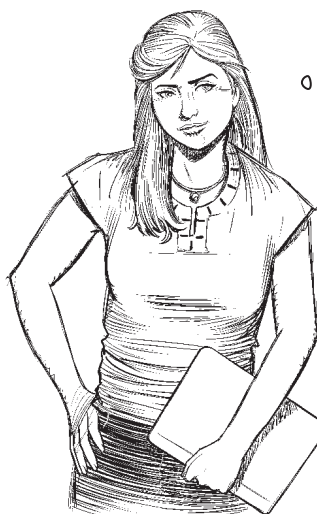
Niektóre decyzje z całą pewnością mają charakter architektoniczny (np. decyzja o wyborze stylu architektury), a inne są wyraźnie związane z projektem (np. zmiana położenia pola na ekranie lub zmiana typu pola w klasie). W rzeczywistości większość napotkanych decyzji mieści się gdzieś pomiędzy tymi dwoma przykładami — gdzieś wewnątrz *spektrum* architektury i projektowania.



Zakreśl wszystko, co Twoim zdaniem mieści się gdzieś pośrodku spektrum pomiędzy architekturą a projektowaniem.

Podział pliku klasy Decyzja o użyciu grafowej bazy danych Wybór frameworka obsługi interfejsu użytkownika
 Wybór frameworka trwałości Podział usługi na części Przeprojektowanie strony internetowej Migracja do mikrousług
 Wybór biblioteki parsującej kod XML

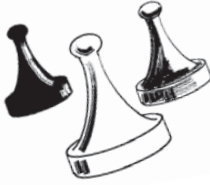
→ Odpowiedź znajdziesz na stronie 33



Dlaczego miałyby mnie obchodzić, gdzie w spektrum między architekturą a projektem leży moja decyzja? Czy to naprawdę ma aż takie znaczenie?

Tak, to ma duże znaczenie. Wiedza o tym, gdzie w spektrum między architekturą a projektem znajduje się dana decyzja, pomaga określić, kto powinien odpowiadać za jej podjęcie. Istnieją pewne decyzje, które powinien podjąć zespół programistów (takie jak projektowanie klas w celu wdrożenia określonej funkcji), z kolei inne powinien podjąć architekt (takie jak wybór najbardziej odpowiedniego stylu architektury dla systemu), a jeszcze inne powinny być podejmowane wspólnie (takie jak dzielenie usług lub ponowne ich scalanie).

W którym miejscu spektrum wypada Twoja decyzja?



Czy decyzja jest strategiczna, czy taktyczna?

Decyzje *strategiczne* są długoterminowe i wpływają na przyszłe działania lub decyzje. Decyzje *taktyczne* są krótkoterminowe i zwykle są niezależne od innych działań lub decyzji (ale mogą być podejmowane w kontekście konkretnej strategii). Na przykład decyzja o wielkości nowego domu wpływa na liczbę pokoi i ich rozmiary, podczas gdy decyzja o wyborze konkretnego oświetlenia nie wpłynie na decyzje dotyczące wielkości stołu w jadalni. Im bardziej strategiczna decyzja, tym bardziej jest ona związana z architekturą.

Czasami wstanie rano z łóżka wymaga wielkiego wysiłku — takie poranki nazwiemy „architektonicznymi”.

Ile wysiłku będą wymagały utworzenie lub zmiana?

Decyzje architektoniczne wymagają większego wysiłku przy budowie lub zmianie, podczas gdy w przypadku decyzji projektowych wysiłek ten jest stosunkowo niewielki. Przykładowo dobudowanie dodatkowego pomieszczenia do domu wymaga dużego nakładu pracy i w związku z tym jest bardziej związane z architekturą, podczas gdy położenie w pokoju dywanu wymaga znacznie mniej wysiłku i w związku z tym jest bardziej związane z projektem.



Czy ta decyzja wiąże się z jakimiś istotnymi kompromisami?

Kompromisy to zalety i wady uwzględniane podczas podejmowania decyzji. Decyzje, które wiążą się ze znaczącymi kompromisami, wymagają znacznie więcej czasu i analiz, a ich charakter jest bardziej architektoniczny. Decyzje wymagające mniej ważkich kompromisów mogą być podejmowane szybciej, przy mniejszej ilości analiz, a zatem mają bardziej projektowy charakter.



Na następnych kilku stronach przedstawimy szczegółowo wszystkie te trzy czynniki.



MOC UMYSŁU

Czy potrafisz znaleźć decyzję, której podjęcie nie wiąże się z żadnymi kompromisami, nieważne jak małymi lub nieistotnymi? Oto wskazówka: jeśli uważasz, że znalazłeś decyzję, która nie wiąże się z jakimkolwiek kompromisem, szukaj dalej.

Strategiczna czy taktyczna

Im bardziej strategiczna jest decyzja, tym bardziej architektoniczna się staje. Jest to ważne rozróżnienie, ponieważ decyzje strategiczne wymagają więcej przemyśleń oraz planowania i są zazwyczaj długoterminowe.



Jak mogę określić, czy dana decyzja jest bardziej strategiczna, czy taktyczna?



Dobre pytanie. Możesz użyć tych trzech pytań, aby określić, czy coś jest bardziej strategiczne, czy taktyczne. Pamiętaj, że im bardziej coś jest strategiczne, tym bardziej chodzi o architekturę.

1. Ile przemyśleń i planowania trzeba włożyć w podjęcie decyzji?

Jeśli podjęcie decyzji zajmuje od kilku minut do godziny, ma ona bardziej taktyczny charakter. Jeśli myślenie i planowanie wymagają kilku dni lub tygodni, jest to prawdopodobnie bardziej strategiczne (a więc bardziej architektoniczne).

2. Ile osób jest zaangażowanych w podejmowanie decyzji?

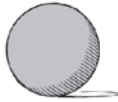
Im więcej zaangażowanych osób, tym bardziej strategiczna decyzja. Decyzja, którą można podjąć samodzielnie lub z kolegą, jest zwykle taktyczna. Decyzja, która wymaga wielu spotkań z wieloma interesariuszami, jest prawdopodobnie bardziej strategiczna.

3. Czy Twoja decyzja wiąże się z długoterminową wizją, czy krótkoterminowym działaniem?

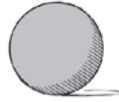
Jeśli podejmujesz szybką decyzję dotyczącą czegoś, co jest tymczasowe lub może się wkrótce zmienić, jest to bardziej taktyczne, a zatem bardziej związane z projektem. I odwrotnie, jeśli jest to decyzja, z którą będziesz żyć przez bardzo długi czas, jest to bardziej strategiczne i bardziej związane z architekturą.

Zaostrz ołówek

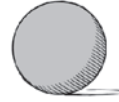
O rany. Zgubiliśmy wszystkie nasze kulki i potrzebujemy Twojej pomocy, aby je zebrać i umieścić z powrotem we właściwych miejscach. Korzystając z trzech pytań z poprzedniej strony, możesz się dowiedzieć, do którego słoika powinny trafić poszczególne kulki.



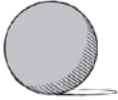
**Wybór języka
programowania dla
nowego projektu**



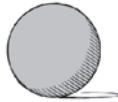
**Decyzja o kupieniu
pierwszego psa**



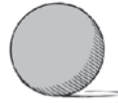
**Wdrażanie
w chmurze lub lokalnie**



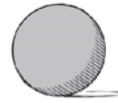
**Przeprojektowanie
interfejsu
użytkownika**



**Migracja systemu
do mikrouslug**



**Wybór biblioteki
parsującej**



**Zastosowanie
wzorca
projektowego**



Strategiczna



**Gdzieś
pomiędzy**



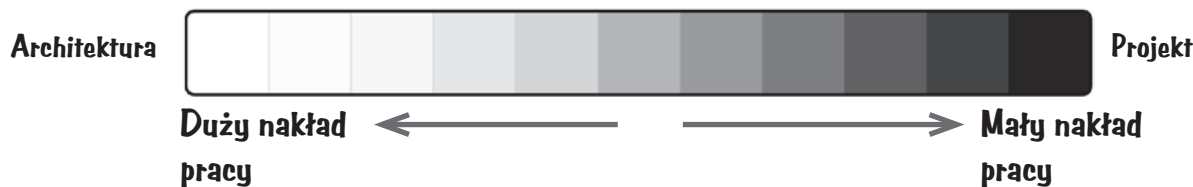
Taktyczna

→ Rozwiązanie znajdziesz na stronie 34

Wysoki czy niski nakład pracy

Znany architekt oprogramowania i autor Martin Fowler napisał kiedyś, że „architektura oprogramowania to rzeczy, które trudno zmienić”. Możesz użyć definicji Martina, aby określić, w którym miejscu spektrum znajduje się Twoja decyzja. Im trudniej jest coś później zmienić, tym bardziej jest to związane z architekturą. I odwrotnie, im łatwiej daną rzecz można później zmienić, tym bardziej jest prawdopodobne, że jest związana z projektowaniem.

Strona Martina Fowlera (<https://martinfowler.com/architecture>) zawiera wiele przydatnych informacji na temat architektury.



Założmy, że planujesz przejście z jednego stylu architektury na inny; powiedzmy, z tradycyjnej architektury wielowarstwowej na mikrousługi. Taka migracja jest dość trudna, a jej przeprowadzenie wymaga wiele czasu i dużego nakładu pracy. Ponieważ nakład pracy jest wysoki, taką migrację należałoby umiejscowić na dalekim końcu spektrum architektury.

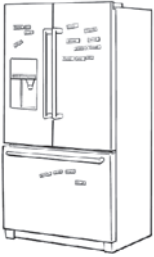


Rany, to będzie wymagało sporo wysiłku. Modyfikowanie rzeczy, które są architektoniczne, jest trudne.

Założmy teraz, że zmieniasz kolejność pól na ekranie interfejsu użytkownika. To zadanie wymaga stosunkowo mało wysiłku, więc znajduje się na drugim końcu spektrum projektowania.

Zmiana układu pól na stronie internetowej jest bardziej związana z wyglądem niż strukturą — to kolejny powód, dla którego można to uznać za projekt.





Magnesiki z kodem

Wszystkie magnesiki z naszej listy rzeczy do zrobienia były porządnie ułożone od dużego do małego nakładu pracy, ale w jakiś sposób spadły na podłogę i się pomieszały. Czy możesz pomóc nam ułożyć je z powrotem we właściwej kolejności w oparciu o nakład pracy konieczny do wykonania każdej modyfikacji?

Narysuj strzałki, które pozwolą rozmieścić poszczególne zadania z listy rzeczy do zrobienia na skali nakładu pracy — te bardziej pracochłonne wyżej, a te mniej niżej.

Rozwiązywanie konfliktów
scalania w systemie Git

Zmiana frameworka obsługi
interfejsu użytkownika na inny

Migracja systemu do
środowiska chmurowego

Podjęcie decyzji o zakupie
musztardy

Zmiana nazwy metody lub
funkcji

Rozbicie pojedynczej usługi na
kilka odrębnych

Przejęcie z relacyjnej bazy
danych na grafową

Podział pliku klasy

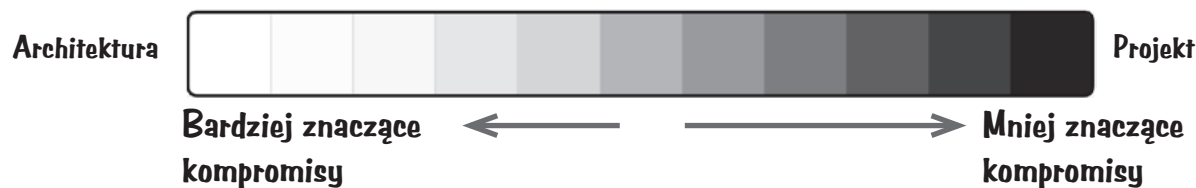
**Duży
wysiętek**

Mały wysiętek

→ Odpowiedź znajdziesz na stronie 35

Kompromisy znaczące i mniej znaczące

Niektóre podejmowane decyzje mogą się wiązać ze znaczącymi kompromisami, takimi jak wybór miasta, w którym chcesz mieszkać. W przypadku innych decyzji kompromisy mogą mieć mniej poważne znaczenie. Przykładem takiej decyzji może być choćby wybór koloru dywanu w salonie. Poziom znaczenia kompromisów związanych z podejmowaniem danej decyzji może pomóc w określeniu, czy decyzja ta dotyczy bardziej architektury, czy projektu. Im bardziej znaczące kompromisy, tym bardziej chodzi o architekturę; im mniej znaczące kompromisy, tym bardziej chodzi o projekt.



- + Skalowalność
- + Zwinność
- + Elastyczność
- + Odporność na błędy



- Koszt
- Złożoność
- Wydajność
- Przepływ pracy

Wow, to kilka poważnych kompromisów do rozważenia. Chodzi bardziej o architekturę.



- + Łatwość utrzymania
- + Czytelność



- Większa liczba klas

Ten kompromis nie jest tak znaczący, co sprawia, że decyzja ta jest bardziej związana z projektem.



Ćwiczenie

Decyzje, decyzje, decyzje. Jak możemy podjąć wszystkie te decyzje? Jedną z rzeczy, które naszym zdaniem mogą pomóc, jest zidentyfikowanie decyzji, które wiążą się z istotnymi kompromisami, ponieważ będą one wymagały więcej myślenia i zajmą więcej czasu. Czy możesz nam pomóc, określając, które z poniższych decyzji wiążą się z istotnymi kompromisami, a które nie?

Czy ten kompromis jest istotny?

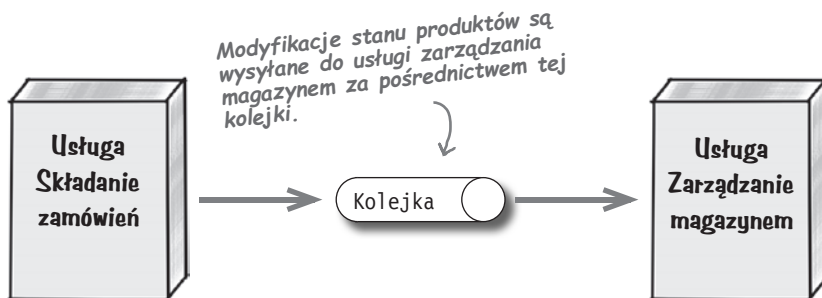
- | | | |
|------------------------------|------------------------------|---|
| ☐ Tak | ☐ Nie | Wybieranie ubrań do pracy na dziś |
| ☐ Tak | ☐ Nie | Wybór wdrożenia w chmurze lub na miejscu |
| ☐ Tak | ☐ Nie | Wybór frameworka interfejsu użytkownika |
| ☐ Tak | ☐ Nie | Nazywanie zmiennej w pliku klasy |
| ☐ Tak | ☐ Nie | Wybór między lodami waniliowymi a czekoladowymi |
| ☐ Tak | ☐ Nie | Decyzja o wyborze stylu architektonicznego |
| ☐ Tak | ☐ Nie | Wybór między REST a przesyłaniem komunikatów |
| ☐ Tak | ☐ Nie | Używanie pełnych danych lub tylko kluczy dla ładunku wiadomości |
| ☐ Tak | ☐ Nie | Wybór biblioteki do parsowania kodu XML |
| ☐ Tak | ☐ Nie | Podejmowanie decyzji o rozdzieleniu usługi |
| ☐ Tak | ☐ Nie | Wybór między transakcjami atomowymi a rozproszonymi |
| ☐ Tak | ☐ Nie | Podejmowanie decyzji, czy wyjść dziś na kolację, czy nie |

→ Odpowiedź znajdziesz na stronie 36

Łączenie wszystkiego w całość

Teraz nadszedł czas, aby wykorzystać wszystkie opisane wcześniej czynniki i dowiedzieć się, czy decyzja dotyczy bardziej architektury, czy raczej projektu. Dzięki temu zespoły programistów wiedzą, kiedy współpracować z architektem, a kiedy podejmować decyzje samodzielnie.

Założmy, że chcąc zwiększyć szybkość reakcji systemu podczas składania przez klientów zamówień, zdecydowałeś się użyć asynchronicznego przesyłania komunikatów między usługą składania zamówień a usługą zarządzania magazynem. W końcu dlaczego klient miałby czekać, aż firma sprawdzi i ewentualnie zmodyfikuje stan magazynu? Zobaczmy, czy możemy określić, w którym miejscu spektrum leży ta decyzja.



Waga kompromisów

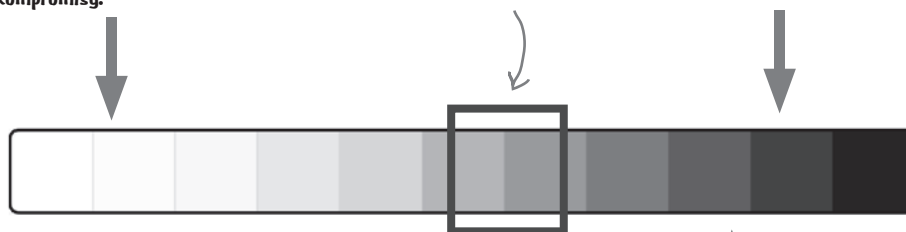
Korzystanie z kolejki zwiększy szybkość reakcji podczas składania zamówienia, ale zapasy mogą nie zostać zaktualizowane w odpowiednim czasie, co może spowodować powstanie zaległości w zamówieniach. Są to dość znaczące kompromisy.

Znaczące kompromisy przybliżają tę decyzję do architektury.

Strategiczna czy taktyczna?

Niewiele osób musi być zaangażowanych w tę decyzję i nie wymaga ona długoterminowego planowania, więc jest bardziej taktyczna.

Architektura



Projekt

Biorąc pod uwagę średnią ze wszystkich trzech czynników, decyzja znajduje się mniej więcej w tym miejscu, co oznacza, że ma ona pewne aspekty architektoniczne i prawdopodobnie należy się skonsultować z architektem lub zaangażować go do zespołu. Potrzebowaliśmy wszystkich trzech czynników, aby określić, czy ta decyzja była bardziej związana z architekturą, czy projektem.

Poziom wysiłek

Wysłanie wiadomości do innej usługi nie wymaga wiele wysiłku. To dość standardowa czynność.

Udało Ci się!

Gratulacje! Przebrnąłeś przez pierwszą część swojej podróży ku zrozumieniu architektury oprogramowania. Zanim jednak zakasziesz rękawy, by zagłębić się w kolejnych rozdziałach, oto mały quiz, który pozwoli Ci sprawdzić swoją dotychczasową wiedzę. Zakreśl kółkiem, czy każde z poniższych stwierdzeń jest prawdziwe, czy fałszywe.

PRAWDA CZY FAŁSZ?

Prawda	Fałsz	Projekt jest jak struktura domu (ściany, dach, układ itd.), a architekturę oprogramowania można by raczej porównać z meblami i dekoracjami.
Prawda	Fałsz	Większość decyzji dotyczy wyłącznie architektury lub projektu. Bardzo niewiele z nich znajduje się w spektrum pomiędzy architekturą a projektem.
Prawda	Fałsz	Im bardziej strategiczna decyzja, tym bardziej chodzi o architekturę; im bardziej taktyczna, tym bardziej chodzi o projekt.
Prawda	Fałsz	Im więcej wysiłku wymaga wdrożenie lub zmiana decyzji, tym bardziej chodzi o projekt; im mniej wysiłku, tym bardziej chodzi o architekturę.
Prawda	Fałsz	<i>Kompromisy</i> to plusy i minusy danej decyzji lub zadania. Im bardziej znaczące stają się kompromisy, tym bardziej chodzi o architekturę.

—————> Odpowiedź znajdziesz na stronie 37

KLUCZOWE ZAGADNIENIA

- Architektura oprogramowania jest mniej związana z wyglądem, a bardziej ze strukturą, podczas gdy projekt jest bardziej związany z wyglądem, a mniej ze strukturą.
- Aby zrozumieć i opisać architekturę oprogramowania, należy użyć czterech wymiarów: cech architektury, decyzji architektonicznych, komponentów logicznych i stylu architektury.
- Cechy architektury stanowią fundamentalne aspekty architektury oprogramowania. Musisz wiedzieć, które z tych cech są najważniejsze dla Twojego konkretnego systemu, abyś mógł analizować kompromisy i podejmować właściwe decyzje architektoniczne.
- Decyzje architektoniczne służą jako drogowskazy, które pomagają zespołom programistów zrozumieć ograniczenia i warunki architektury.
- Komponenty logiczne wyróżnione w konkretnym rozwiązaniu architektury oprogramowania tworzą bloki konstrukcyjne systemu. Reprezentują one rzeczy, które system robi, i są implementowane za pomocą plików klas lub kodu źródłowego.
- Podobnie jak w przypadku domów, także w przypadku oprogramowania istnieje wiele różnych stylów architektury. Każdy z tych stylów wspiera określony zestaw cech architektury, dlatego tak istotne jest, aby upewnić się, że dla swojego systemu wybrałeś odpowiedni styl (lub ich kombinację).
- Ważne jest, aby wiedzieć, czy decyzja dotyczy architektury, czy projektu, ponieważ pomaga to określić, kto powinien być za nią odpowiedzialny i na ile jest istotna.



Ćwiczenie

Rozwiązanie

Ze strony 2

Kolejną przydatną i popularną metaforą architektury oprogramowania jest ogrodnictwo. W pustym obszarze w dolnej części tej ramki spróbuj opisać, w jaki sposób planowanie ogrodu może się odnosić do architektury oprogramowania.

Ogólny układ ogrodu można porównać do stylu architektury, podczas gdy każda grupa podobnych roślin (według typu lub koloru) może reprezentować komponenty architektury. Poszczególne rośliny w grupie reprezentują pliki klas implementujące te komponenty.

Ogrody podlegają wpływom pogody w taki sam sposób, w jaki na architekturę oprogramowania wpływają zmiany w technologii, platformach, środowisku wdrożeniowym itp. Ponadto jeśli nie zwracasz uwagi na ogród, wyrastają w nim chwasty — podobny „rozkład strukturalny” zachodzi w Twojej architekturze.

Ze strony 3



Zaostrz ołówek

Rozwiązanie

Jakie cechy domu, strukturalne i związane z jego architekturą, jesteś w stanie wymienić?

Rozmiar i kształt kuchni
(kto nie narzeka na to,
że jego kuchnia jest mała?).

Liczba pięter
(wraz z wiekiem
schody mogą
stanowić problem).

Gdzie znajdują się drzwi
wejściowe i czy wejście
jest dostępne dla wózków
inwalidzkich.

Rozmiar szafy w sypialni
(jeśli masz dużo ubrań).

Wysokość sufitu
(zwłaszcza jeśli
jesteś bardzo wysoki).

Liczba łazienek (dodanie nowej
łazienki jest naprawdę trudne).

Strych do przechowywania
wszystkich rzeczy, których
nigdy nie używasz.

Zewnętrzny taras
lub patio (oczywiście,
chyba że mieszkasz
w Arktyce).



Ćwiczenie

Rozwiązanie:

Ze strony 6

Zaznacz to, co Twoim zdaniem można uznać za cechy architektury — coś, co jest wspierane przez *strukturę* systemu oprogramowania.

☐ Zmiana rozmiaru czcionki w oknie prezentującym interfejs użytkownika

☒ Szybkie wprowadzanie zmian

W przypadku architektury oprogramowania ta cecha jest nazywana zwinnością.

☒ Obsługa tysięcy jednocześnie pracujących użytkowników

Ta cecha jest nazywana elastycznością.

☐ Szyfrowanie przechowywanych w bazie danych haseł użytkowników

Ta cecha jest określana jako współdziałanie.

☒ Interakcja z wieloma systemami zewnętrznymi w celu realizacji potrzeb biznesowych

Kto co robi?

Rozwiązanie

Ze strony 7

Oto szansa na sprawdzenie, ile już wiesz o wielu typowych cechach architektury. Czy potrafisz dopasować każdą z cech przedstawionych po lewej stronie do definicji podanej po prawej? Zauważysz, że definicji jest więcej niż cech, więc bądź ostrożny — nie wszystkie definicje mają pasującą nazwę.

Rozszerzalność

To zrobiliśmy za Ciebie.

Zwinność

Współdziałanie

Odporność na błędy

Wykonalność

Uwzględnienie przy dokonywaniu wyborów architektonicznych ram czasowych, budżetów i umiejętności programistów.

Zdolność systemu do zachowania prawidłowego funkcjonowania innych jego części w przypadku wystąpienia błędów krytycznych.

Łatwość, z jaką system może zostać rozbudowany w celu obsługi dodatkowych możliwości i funkcjonalności.

Czas potrzebny na uzyskanie odpowiedzi od użytkownika.

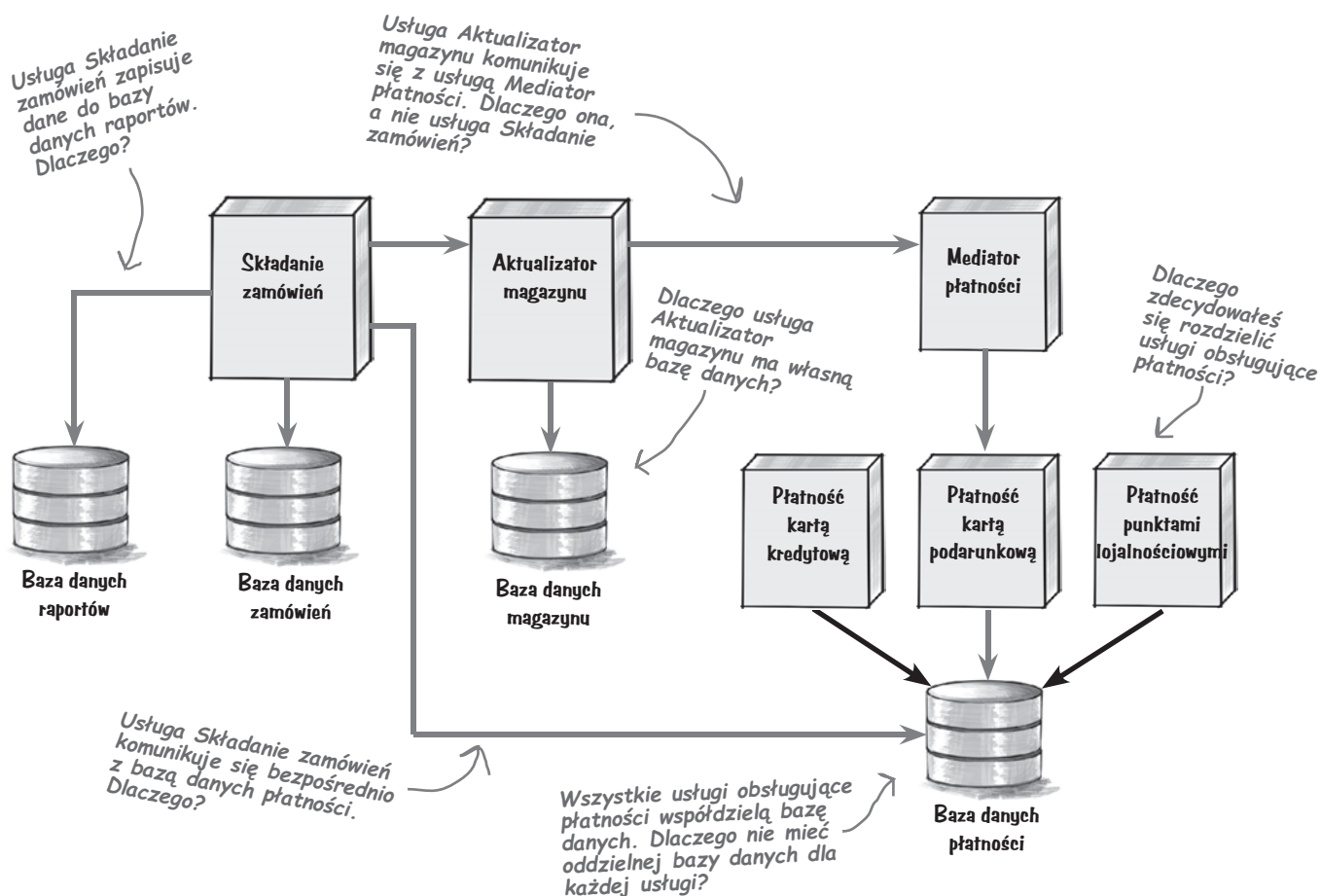
Zdolność systemu do szybkiego reagowania na zmiany (funkcja łatwości utrzymania, testowania i wdrażania).

Zdolność systemu do współdziałania i interakcji z innymi systemami w celu realizacji potrzeb biznesowych.

Bądź architektem. Rozwiązanie



Twoim zadaniem jest wcielenie się w rolę architekta i zidentyfikowanie na poniższym diagramie jak najwięcej decyzji architektonicznych. Narysuj kółko wokół wszystkiego, co Twoim zdaniem może być decyzją architektoniczną, i napisz, jaka to może być decyzja.



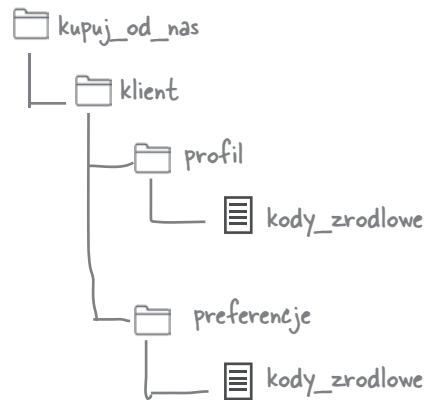
Zaostrz ołówek

Rozwiązanie

Właśnie utworzyłeś następujące dwa komponenty dla nowego systemu, a Twój zespół programistów chce rozpocząć pisanie kodu klas, aby je zaimplementować. Czy możesz utworzyć dla nich strukturę katalogów, aby zespół mógł rozpocząć pracę nad kodem?

**Profil
klienta**

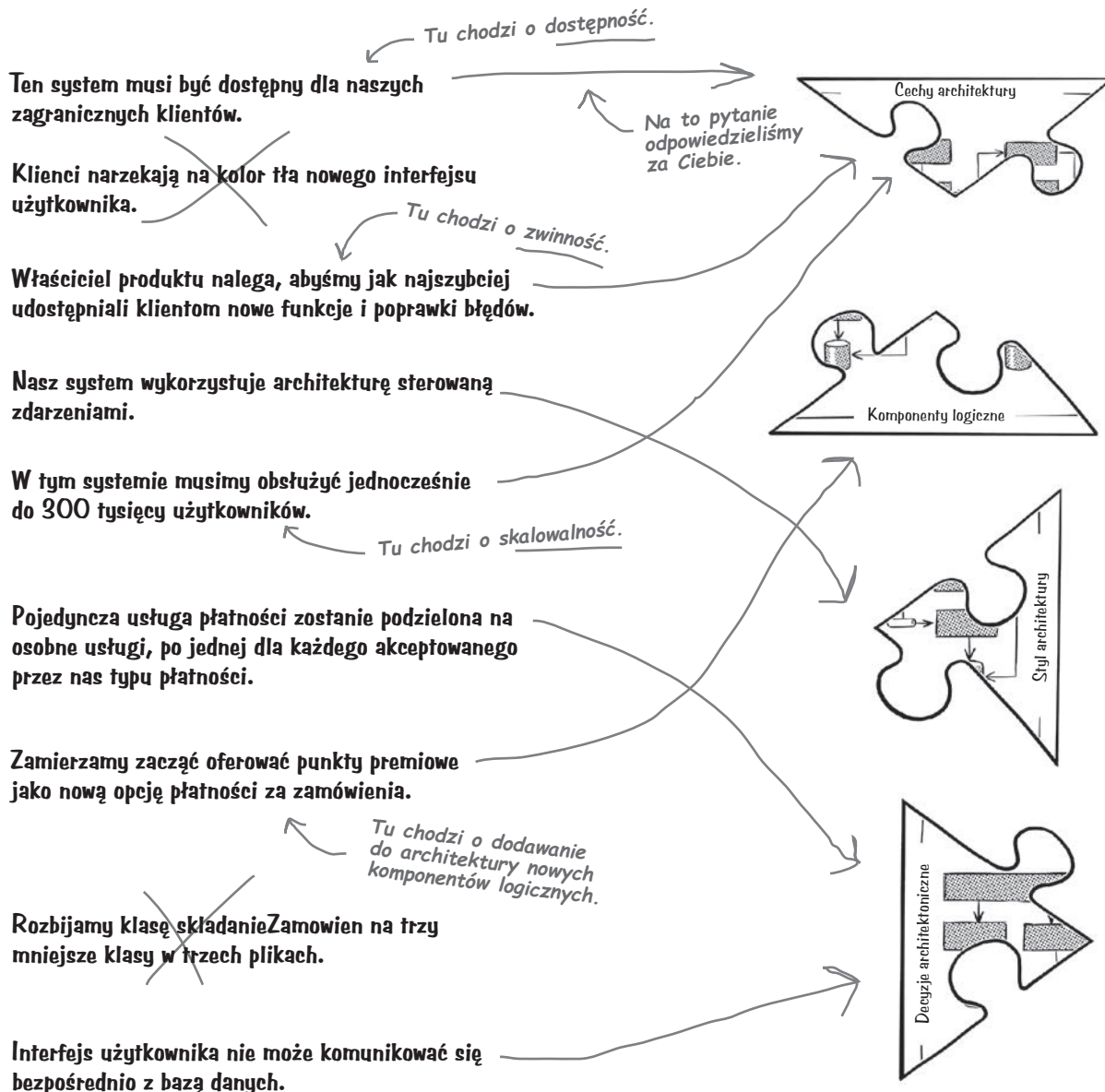
**Preferencje
klienta**



Kto co robi?

Rozwiązanie

Oto szansa na sprawdzenie, ile już wiesz o wielu typowych cechach architektury. Czy potrafisz dopasować każdą z cech przedstawionych po lewej stronie do definicji podanej po prawej? Zauważysz, że definicji jest więcej niż cech, więc bądź ostrożny — nie wszystkie definicje mają pasującą nazwę.





Ćwiczenie

Rozwiązanie

Zaznacz wszystko, co powinno być zawarte w diagramie narysowanym z *punktu widzenia architektury*.

- ☒ Jak usługi komunikują się ze sobą
- ☐ Platforma i język, w którym zaimplementowane są usługi
- ☒ Które usługi mają dostęp do których baz danych
- ☒ Ile jest usług i baz danych

To, w jaki sposób coś należy zaimplementować, jest zagadnieniem związanym z projektem.



Zaostrz ołówek

Rozwiązanie

Zakreśl wszystko, co Twoim zdaniem mieści się gdzieś pośrodku spektrum *pomiędzy* architekturą a projektowaniem.

To jest związane z projektem.

Podział pliku klasy

Decyzja o użyciu grafowej bazy danych

Wybór frameworka trwałości

Podział usługi na części

Przeprojektowanie strony internetowej

Te zagadnienia są związane z projektem.

Wybór frameworka obsługi interfejsu użytkownika

To jest związane z architekturą.

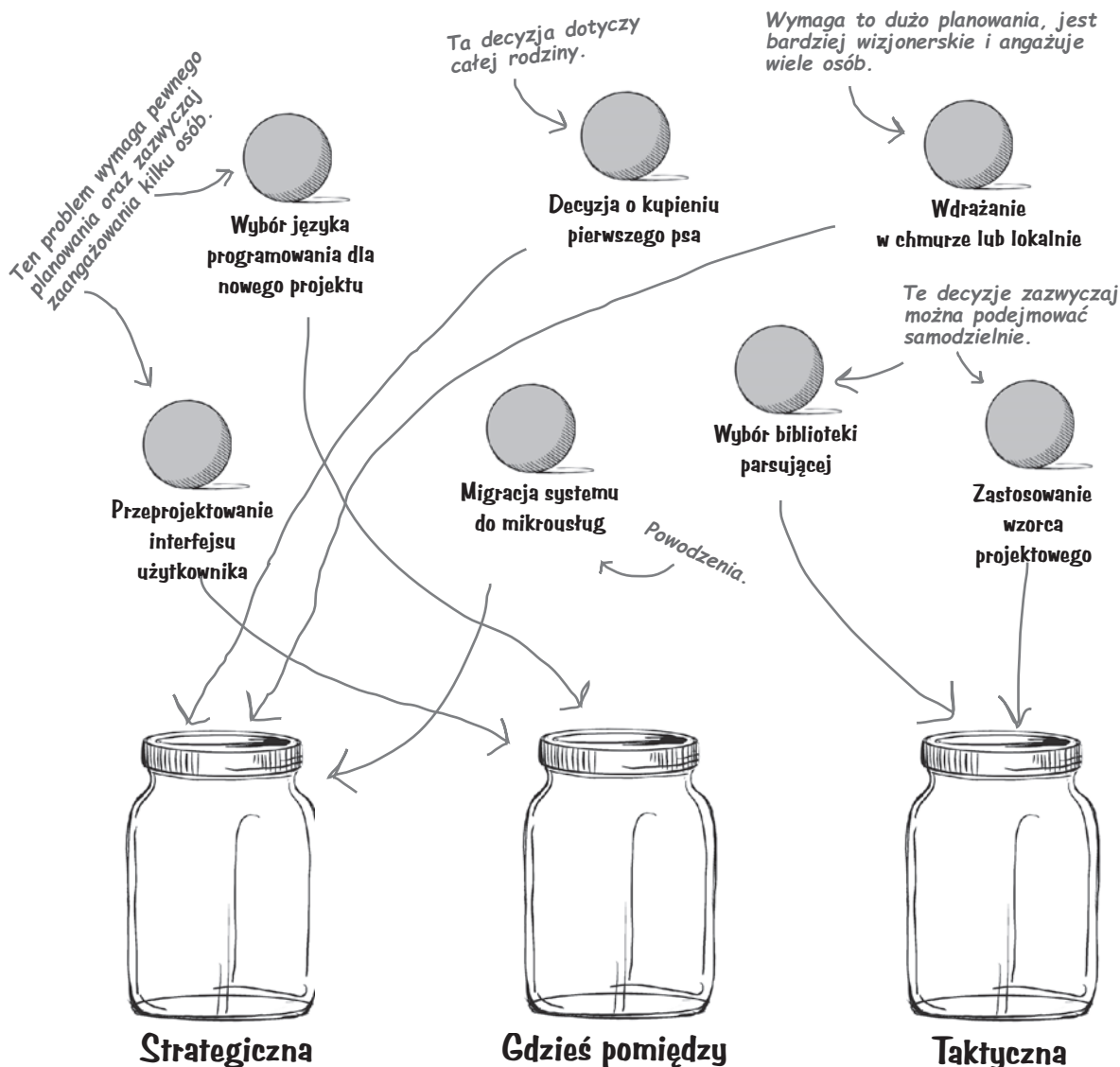
Migracja do mikrousług

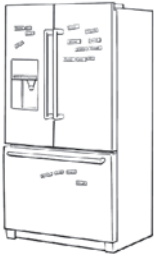
Wybór biblioteki parsującej kod XML

Zaostrz ołówek

Rozwiązanie

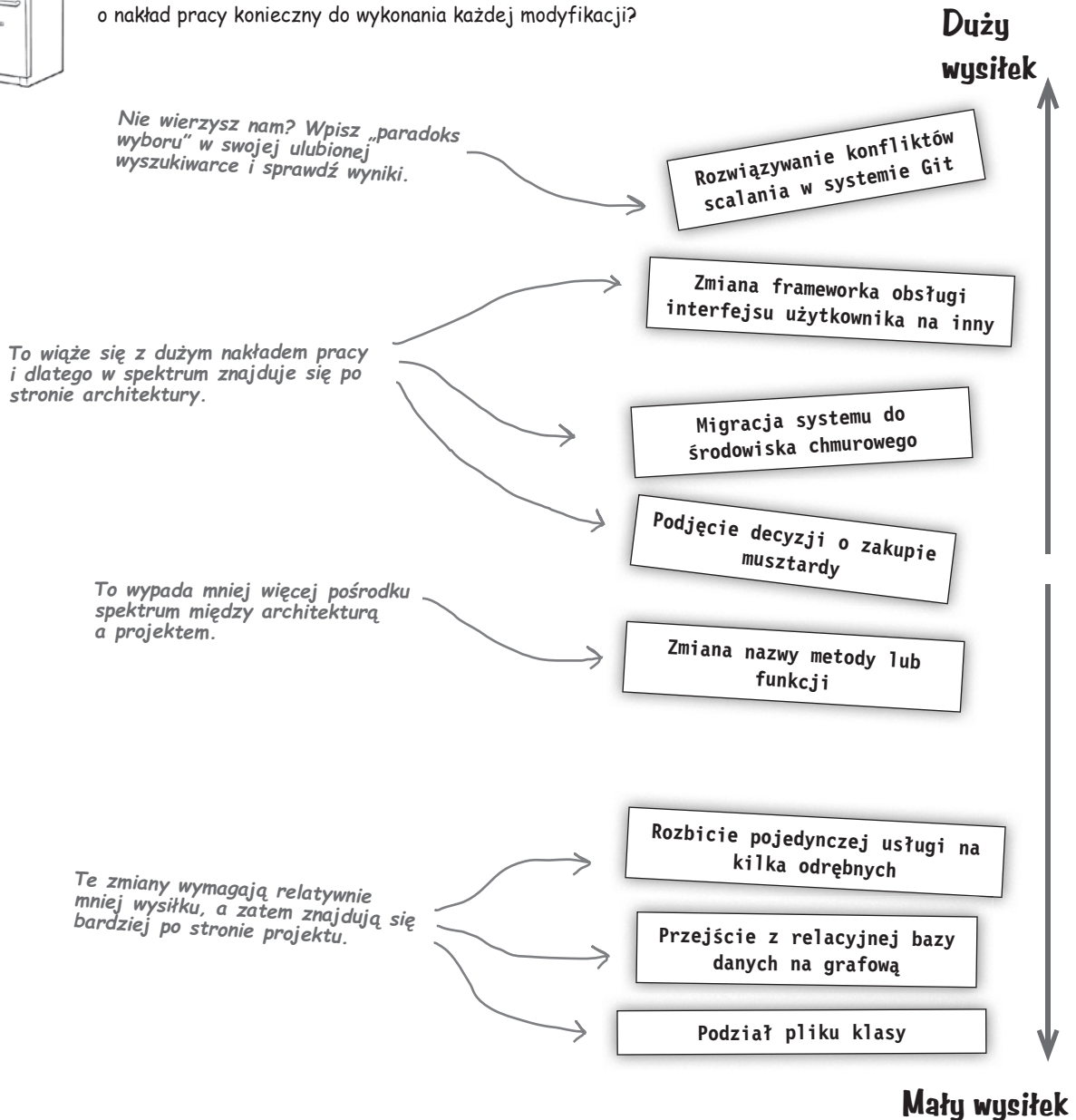
O rany. Zgubiliśmy wszystkie nasze kulki i potrzebujemy Twojej pomocy, aby je zebrać i umieścić z powrotem we właściwych miejscach. Korzystając z trzech pytań z poprzedniej strony, możesz się dowiedzieć, do którego stoika powinny trafić poszczególne kulki.





Magnesiki z kodem. Rozwiązanie

Wszystkie magnesiki z naszej listy rzeczy do zrobienia były porządnie ułożone od dużego do małego nakładu pracy, ale w jakiś sposób spadły na podłogę i się pomieszały. Czy możesz pomóc nam ułożyć je z powrotem we właściwej kolejności w oparciu o nakład pracy konieczny do wykonania każdej modyfikacji?





Ćwiczenie

Rozwiązanie

Ze strony 25

Decyzje, decyzje, decyzje. Jak możemy podjąć wszystkie te decyzje? Jedną z rzeczy, które naszym zdaniem mogą pomóc, jest zidentyfikowanie decyzji, które wiążą się z istotnymi kompromisami, ponieważ będą one wymagały więcej myślenia i zajmą więcej czasu. Czy możesz nam pomóc, określając, które z poniższych decyzji wiążą się z istotnymi kompromisami, a które nie?

Czy ten kompromis jest istotny?

Okej, być może czasami to jest trudna decyzja.

- | | | | |
|---|---|---|---|
| ☐ Tak | ☒ Nie | Wybieranie ubrań do pracy na dziś | |
| ☒ Tak | ☐ Nie | Wybór wdrożenia w chmurze lub na miejscu | |
| ☐ Tak | ☒ Nie | Wybór frameworka interfejsu użytkownika | ← Z pewnością występują tu pewne kompromisy, więc ta sprawa może pójść w obie strony. |
| ☐ Tak | ☒ Nie | Nazywanie zmiennej w pliku klasy | |
| ☐ Tak | ☒ Nie | Wybór między lodami waniliowymi a czekoladowymi | |
| ☒ Tak | ☐ Nie | Decyzja o wyborze stylu architektonicznego | ← To może mieć wpływ na skalowalność, wydajność i ogólną łatwość konserwacji. |
| ☒ Tak | ☐ Nie | Wybór między REST a przesyłaniem komunikatów | ← |
| ☒ Tak | ☐ Nie | Używanie pełnych danych lub tylko kluczy dla ładunku wiadomości | ← |
| ☐ Tak | ☒ Nie | Wybór biblioteki do parsowania kodu XML | |
| ☒ Tak | ☐ Nie | Podjęcie decyzji o rozdzieleniu usługi | |
| ☒ Tak | ☐ Nie | Wybór między transakcjami atomowymi a rozproszonymi | ← |
| ☐ Tak | ☒ Nie | Podjęcie decyzji, czy wyjść dziś na kolację, czy nie | ← |

Czy już zaczynasz być głodny?

Może to mieć wpływ na integralność i spójność danych, ale także na skalowalność i wydajność.

PRAWDA CZY FAŁSZ?**ROZWIĄZANIE**

Prawda **Fałsz** Projekt jest jak struktura domu (ściany, dach, układ itd.), a architekturę oprogramowania można by raczej porównać z meblami i dekoracjami.

Jest dokładnie na odwrót.

Prawda **Fałsz** Większość decyzji dotyczy wyłącznie architektury lub projektu. Bardzo niewiele z nich znajduje się w spektrum pomiędzy architekturą a projektem.

Większość decyzji mieści się w spektrum między architekturą a projektem.

Prawda Fałsz Im bardziej strategiczna decyzja, tym bardziej chodzi o architekturę; im bardziej taktyczna, tym bardziej chodzi o projekt.

Prawda **Fałsz** Im więcej wysiłku wymaga wdrożenie lub zmiana decyzji, tym bardziej chodzi o projekt; im mniej wysiłku, tym bardziej chodzi o architekturę.

Jest dokładnie na odwrót.

Prawda Fałsz *Kompromisy* to plusy i minusy danej decyzji lub zadania. Im bardziej znaczące stają się kompromisy, tym bardziej chodzi o architekturę.

Skorowidz

A

ADR, Architectural Decision
Record, 93

akcja, 128, 131, 400

aktor, 128, 131, 400

analiza

cech architektury, 136, 138

kompromisów, 83, 110, 247

kolejki, 84

tematy, 85

kosztów i korzyści, 88

ról i odpowiedzialności, 134

aplikacja podróznicza

cechy architektury, 265–267

diagram architektury, 274

dokumentacja, 272

komponenty logiczne, 265, 268, 279

planowanie architektury, 264

przepływ pracy użytkownika, 263

styl architektury, 270

tworzenie, 261

wymagania, 262

architekt oprogramowania

architektoniczne kata, 418

oczekiwania, 410

piramida wiedzy, 416

pisanie kodu, 408

umiejętności miękkie, 412

architektura

dwuwarstwowa, 189, 190, 200

fizyczna, 121, 122, 150, 190

analiza kompromisów, 247

tworzenie diagramu, 396, 398

hybrydowa, 276, 364, 398

logiczna, 121, 122, 150

planowanie, 123

mikrojądra, 233, 237–239, 256,

270, 280

diagram, 286

jądro, 237

monolityczna, 244, 256, 259

oceny, 254

rozproszona, 244, 259

stosowanie, 256

wady, 253, 259, 280

wtyczki, 237

zalety, 252, 259, 280

mikrouslug, 12, 239, 291, 320,
371, 379

cechy architektury, 392

choreografia, 308, 309, 312,
314, 328

diagram, 405

fizyczne konteksty powiązane,
292, 360

oceny, 318

orkiestracja, 308–311, 328

poziom szczegółowości, 294, 295

przepływ pracy, 320, 327

przetwarzanie zdarzeń, 362

przetwarzanie żądań, 362, 371

równoważenie szczegółowości,
300

siły dezintegrujące, 295–297, 320

siły integrujące, 298, 299, 320

sterowana zdarzeniami, 364

styl komunikacji, 362

szczególność, 320

szczególność danych, 361

szczególność usług, 361

wady, 304, 305, 317, 403

własności danych, 292

współdzielenie funkcjonalności,
303

współdzielona biblioteka,

305–307, 320, 326

współdzielona usługa, 304, 306,
307, 320, 326

wydajność, 360

zalety, 304, 305, 316, 403

założenia, 290

zarządzanie przepływem pracy,
308–312

modularnego monolitu, 203, 207,
210, 226, 270

diagram, 285

oceny, 224

podział w oparciu o dziedzinę, 215

wady, 223, 280

zalety, 222, 280

zarządzanie, 226

monolityczna, 45, 161, 170–172,
179, 244

model wdrażania, 164, 166, 167

oprogramowania, 2

planowanie, 123, 261, 264, 386

rozproszona, 45, 164, 168–172

model wdrażania, 168, 169

sterowana zdarzeniami, 12, 331,
333, 379

baza danych, 352

cechy architektury, 392

diagram, 406

fizyczne konteksty

powiązane, 360

komunikacja asynchroniczna,
343, 345, 371

oceny, 368

przepływ pracy, 332, 334, 372

przetwarzanie zdarzeń, 362, 371

przetwarzanie żądań, 362

rozszerzalność

architektoniczna, 342

styl komunikacji, 362

szczególność danych, 361

szczególność usług, 361

topologie baz danych, 371

typy zdarzeń, 340

wady, 367, 403

wydajność, 360

zalety, 366, 403

trójwarstwowa, 189, 190, 200

architektura

- warstwowa, 12, 175, 179, 187, 198, 270
- dzielenie komponentów, 185, 199
- elastyczność, 195
- łatwość testowania, 195
- łatwość wdrażania, 195
- oceny, 196
- skalowalność, 195
- zalety, 188, 196
- warstwowego monolitu, 179
- diagram, 284
- wady, 280
- zalety, 280
- wbudowana/mobilna, 189, 190, 200
- asynchroniczność, 331, 343, 347, 351
- ATAM, Architecture Tradeoff Analysis Method, 88
- atrybuty jakości systemu, 7

B

- baza danych, 179
- modularyzacja, 219, 226
- monolityczna, 352, 371
- oceny topologii, 353
- na usługę, 356, 371
- oceny topologii, 357
- podzielona w oparciu o dziedzinę, 354, 371
- oceny topologii, 355
- błędy asynchroniczne, 350

C

- CA, total afferent coupling, 142, 156
- CBAM, Cost Benefit Analysis Method, 88
- CE, total efferent coupling, 142, 156
- cechy architektury, 4–7, 27, 39, 76, 257, 277, 321, 392
- analiza, 136, 138
- dziedziny problemu, 58
- świadomości środowiskowej, 59
- autoryzacja, 53, 73
- bezpieczeństwo, 42, 51–53, 55
- definiowanie, 43
- dostępność, 53–55, 71
- dostępność techniczna, 52, 136, 137
- dotyczące procesu, 50

- elastyczność, 73, 195
- identyfikacja, 266, 388, 398
- integralność danych, 42
- jawne, 48
- kluczowe, 401
- legalność, 53
- lokalizacja, 51
- łatwość
 - dostosowywania, 73
 - utrzymania, 51, 54, 71
 - wdrażania, 54, 71
- modularność, 54, 71
- możliwości, 66
- kontroli, 42
- odzyskiwania, 52
- rozdzielania, 50
- testowania, 50
- niejawne, 48, 76, 276, 277, 388, 398, 401
- niezawodność, 42, 52
- ograniczanie, 46, 68
- operacyjne, 52
- priorytety, 56, 62
- prywatność, 53
- przekrojowe, 53
- przenośność, 51
- rozszerzalność, 50, 51
- skalowalność, 42, 52, 54, 71, 73, 136, 137
- solidność, 52, 54, 71
- spójność, 42
- strukturalne, 51
- tłumaczenie wymagań, 64
- umiędzynarodawianie, 73
- uwierzytelnianie, 53, 73
- użyteczność, 42, 53
- wiedza o dziedzinie, 59
- wydajność, 52, 136, 137
- złożone, 61
- zwinność, 50
- choreografia, choreography, 308, 309, 312, 314, 328
- CT, total coupling, 142, 156

D

- DDD, domain driven design, 131, 248
- decyzje architektoniczne, 4, 8, 27, 89, 90, 110

- dokumenty ADR, 93–110, 272, 394
- kompromisy, 19
- podejmowanie, 18, 36, 89
- strategiczne, 19, 20
- taktyczne, 19, 20
- uzasadnianie, 102

diagram

- architektury
 - fizycznej, 274, 276, 396
 - mikrojądra, 286
 - mikrousług, 405
 - modularnego monolitu, 285
 - warstwowego monolitu, 284

- klas, 16

diagramy

- techniki tworzenia, 414
- dokumenty ADR, 93–110, 272, 276, 398

sekcja

- Decyzja, 101, 110
- Konsekwencje, 103, 110, 394
- Kontekst, 100, 102, 110
- Nadzór, 105, 110
- Uwagi, 105, 110
- status, 96–99
- RFC, 96, 101
- Zaakceptowany, 96
- Zaproponowany, 96, 101, 282, 283, 404

- Zastąpiony, 97

- tytuł, 95, 110

- dostępność techniczna, availability, 6, 52, 136, 137, 348

- dziedzina, domain, 11, 42, 44, 67, 184, 185

- moduły, 210

- modyfikowanie, 209

- problemu, problem domain, 57, 118

- modyfikowanie, 193, 198

E

- EDA, event-driven architecture, 333
- encje, 130

F

- fizyczne konteksty powiązane, 292, 320, 360

funkcjonalności, 134, 207
 jądra, 258
 systemu, 11

I

identyfikacja początkowych
 komponentów, 124–136
 integralność danych, 6
 interfejs, 243
 API, 221
 użytkownika, 183, 248, 274

J

jądro, 237, 248
 usługa oceny urzędzeń, 241, 251

K

kanal zdarzeń, 333
 kolejki, queues, 82, 84, 114–116
 komponenty logiczne, 4, 10, 27, 43, 66, 117–148
 akt równoważenia, 147
 analiza cech architektury, 136, 138
 analiza ról i odpowiedzialności, 134
 identyfikacja, 123, 124, 268, 276, 390, 398
 oparta na aktorach i akcjach, 128, 131
 oparta na przepływie pracy, 126, 131
 pułapka encji, 130
 interakcje, 402
 początkowe podstawowe, 124–136
 podział, 185
 powiązania, 50, 139
 całkowite, 142
 doprowadzające, 140, 142
 luźne, 147
 odprowadzające, 141, 142
 poziom powiązań, 142
 prawo Demeter, 143, 145
 redukcja powiązań, 146, 157
 ścisłe, 144, 147
 prezentacja, 118
 przypisanie wymagań, 132

spójność, 135
 struktura katalogów, 119
 kompromisy, 19, 24, 37, 77, 110
 analiza, 83
 ATAM, 88
 wersja z kolejkami, 84
 wersja z tematami, 85
 architektury fizycznej, 190
 komunikacji synchronicznej
 i asynchronicznej, 348
 komunikacja
 asynchroniczna, 112, 331, 343–348, 371
 pomiędzy modułami, 226
 pomiędzy usługami, 80
 synchroniczna, 112, 343–345, 349
 typu odpal i zapomnij, 345
 z usługami serwerowymi, 82
 z wtyczkami, 245
 komunikaty, 337, 338, 371
 kontekst, 100, 102
 kontrakty wtyczek, 250
 kontroler, 178

M

mikrojądro, *Patrz* architektura
 mikroądra
 mikrousługi, *Patrz* architektura
 mikrousług
 model, 178
 modelowanie przepływu pracy, 126
 modularne monolity, *Patrz*
 architektura modularnego monolitu
 modularność, 50, 54, 71, 215–217, 226
 modularyzacja
 bazy danych, 219, 226
 kodu, 218
 moduły, 207, 208, 226
 interfejs API, 221
 komponenty, 228
 komunikacja, 226
 modularność, 215–217
 tworzenie, 210, 217
 złączenia, 221, 226
 monolit, *Patrz* architektura
 monolityczna
 monolityczna baza danych, 352, 371
 MVC, Model-Widok-Kontroler, 178

N

nakład pracy, 22, 35
 narzędzie
 ArchUnit, 217
 esLint, 239
 Gradle, 217
 Jenkins, 239
 NVP, minimal viable product, 90

O

obsługa błędów, 349
 asynchronicznych, 350
 odporność na błędy, 7, 29
 orkiestracja, orchestration, 308–310, 314

P

poddziedziny, 229
 podejmowanie decyzji
 architektonicznych, 18, 36, 89
 podprojekty, 217
 powiązanie komponentów, 50, 139
 całkowite, CT, 142
 doprowadzające, CA, 142
 odprowadzające, CE, 142
 doprowadzające, afferent
 coupling, 140
 luźne, 147
 odprowadzające, efferent
 coupling, 141
 pomiar powiązań, 142
 prawo Demeter, 143, 145
 redukcja powiązań, 146, 157
 ścisłe, 144, 147
 poziom
 planowania, 20
 wysiłku, 22
 prawo
 architektury oprogramowania, 86, 92, 110
 Demeter, 143, 146
 stosowanie, 145
 prezentacja, 179, 183
 projekt, 18, 37
 projektowanie, 15
 na wyrost, overengineering, 46, 67
 oparte na dziedziniu, DDD, 131

przepływ pracy, workflow, 126, 179,
183, 308, 309, 320, 332, 372, 385
przestrzeń nazw, 182, 212
pseudokod, 183

R

responsywność, 333
REST, 256, 343
RFC, Request For Comments, 96
role i odpowiedzialność, 134
rozszerzalność, 7, 29, 50, 51
architektoniczna, architectural
extensibility, 342

S

skalowalność, 6, 52, 71, 136, 195, 333
struktura
architektury, 45
katalogów, 118, 119, 182
systemu, 17
style architektury, 4, 12, 27, 159–161,
172
architektura warstwowa, 270, 280
mikrojądro, 270, 280, 329
mikrouługi, 287
modularny monolit, 203, 270, 280
typy
modeli wdrożenia, 164
podziałów, 162
wybór, 270, 392
synergia, 47
szczegółowość, granularity, 295
dezintegratory, granularity
disintegrators, 295, 296
danych, 361
integratory, granularity
integrators, 295, 298
usług, 361

T

tematy, topics, 82, 85
trwałość, 179, 183

tworzenie
architektury, 123, 261, 264, 386
izolowanych modułów, 217

U

UML, Unified Modeling Language, 16
usługa
analityczna, 80
handlowa, 78
orkiestracji, orchestration
service, 310
powiadomień, 80
usługi
dodawanie warstwy usług, 192
serwerowe, 82
własne bazy danych, 356
współdzielone, 320

W

warstwa, 185, 198
prezentacji, 179, 183
przepływu pracy, 179, 183
trwałości, 179, 183
usług, 192
warstwy wzorca MVC, 179
wdrażanie
model monolityczny, 164
wady, 167, 174
zalety, 166, 174
model rozproszony, 164
wady, 169, 174
zalety, 168, 174
widok, 178
współdziałanie, 7, 29
wtyczki, 237, 238, 246, 247
hermetyzowane, 243
implementacja interfejsu, 243, 251
komunikacja z systemem, 245,
249, 250
kontrakty, 249, 250
rozproszone, 242–245
stosowanie, 239

wywołania, 256
wydajność, 6, 136, 137
wykonalność, 7, 29
wymagania, 40, 60, 398
dla aplikacji
do śledzenia wydatków, 229
do zarządzania podróżami, 262
internetowej restauracji, 176
obsługującej recykling, 234
niefunkcjonalne, 7, 56
przekładanie na cechy, 64
przekrojowe, 56
przypisywanie, 132
systemu
monitorowania, 288
zamówień internetowych, 334
przeprowadzania testów, 388
wymiały architektury
oprogramowania, 4
cechy architektury, 6
decyzje architektoniczne, 8
komponenty logiczne, 4, 10
style architektury, 4, 12
wzorzec projektowy, 178
Dekorator, 248
MVC, 178, 198
warstwy, 179

Z

zarządzanie przepływem pracy, 308,
309
zdarzenia, 331, 333, 336–338
inicjujące, initiating events, 340, 371
pochodne, derived events, 340, 371
zespoły przekrojowe, 210
złączenia modułów, 221, 226
zwinność, agility, 7, 12, 29, 32, 50
zwrot z inwestycji, ROI, 88

PROGRAM PARTNERSKI

— GRUPY HELION —

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

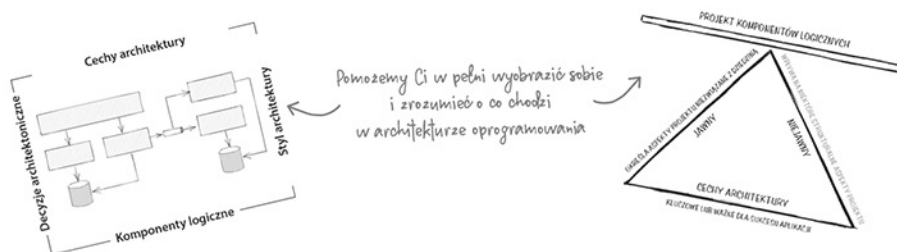
Zmień swoją stronę WWW w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA
Helion 

Jeśli zależy Ci na sukcesie systemu, który budujesz, musisz zadbać o jego architekturę! Dobre praktyki architektoniczne poprawiają jakość oprogramowania, które skutecznie dostosowuje się do wymagań, nawet podczas bezustannych zmian środowisk biznesowych i technicznych. Jednak architektura oprogramowania jest trudnym zagadnieniem, więc Twój mózg może się starać Cię przekonać, że się jej nie nauczysz. Zatem pora go oszukać!



Ta fantastyczna książka, pełna zabawnej narracji i praktycznych przykładów, nauczy Cię myślenia architektonicznego. Została ona, podobnie jak inne pozycje z serii **Rusz głową!**, przygotowana na bazie odkryć nauk poznawczych i neurofizjologii. Właśnie dzięki temu zaangażujesz swój mózg, użyjesz wielu zmysłów i niepostrzeżenie zrozumiesz dwa prawa architektury oprogramowania i cztery opisujące ją wymiary. Mistrzowsko opanujesz cechy i style architektury, a także nauczysz się określać logiczne komponenty systemów. Efekt? Zdobędziesz świetną orientację w świecie architektury oprogramowania. A wszystkiego nauczysz się, rozwiązując łamigłówki, wykonując praktyczne ćwiczenia, tworząc architektury – i wybuchając głośnym śmiechem!

Autorzy w mistrzowski sposób przekształcają złożone pojęcia w łatwo przyswajalne informacje!

James Erler
inżynier oprogramowania
wbudowanego Medtronic

Raju Gandhi jest architektem, konsultantem, autorem i prele-gentem. Biegle posługuje się różnymi językami programowania i paradygmatami. Wierzy w prostotę.

Mark Richards jest doświad-czonym architektem oprogra-mowania, autorem licznych książek technicznych i filmów instruktażowych, a także trenerem i prele-gentem.

Neal Ford jest dyrektorem, architektem oprogramowania i twórcą memów w ThoughtWorks. Jest uznanym na całym świecie ekspertem w dziedzinie rozwoju i dostarczania oprogramowania.

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 978-83-289-1567-1



Cena: 129,00 zł