Strategiczne monolity i mikrousługi

Jak napędzać innowacyjność za pomocą przemyślanej architektury

Vaughn Vernon
Tomasz Jaskuła

Przedmowa
Mary Poppendieck

Tytuł oryginału: Strategic Monoliths and Microservices:
Driving Innovation Using Purposeful Architecture

Tłumaczenie: Maksymilian Gutowski

ISBN: 978-83-283-9552-7

Authorized translation from the English language edition, entitled Strategic Monoliths and Microservices: Driving Innovation Using Purposeful Architecture, 1st Edition by Vaughn Vernon; Tomasz Jaskula, published by Pearson Education, Inc, publishing as Addison-Wesley Professional, Copyright © 2022 Pearson Education, Inc

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

Polish language edition published by Helion S.A., Copyright © 2023.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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/strmon
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

Przedmowa 11

Wstęp 15

Podziękowania 21

O autorach 25

Część I. Strategiczne uczenie się poprzez eksperymenty na potrzeby transformacji 27

Streszczenie 29

Rozdział 1. Cele biznesowe i transformacja cyfrowa 33
- Transformacja cyfrowa — co jest jej celem? 34
  - Architektura oprogramowania — szybki przegląd 36
- Dlaczego oprogramowanie się nie sprawdza? 37
  - Metafora długu 38
  - Entropia oprogramowania 39
  - Wielka kula błota 39
  - Bieżący przykład 40
- Twoje przedsiębiorstwo a prawo Conwaya 44
  - Komunikacja dotyczy wiedzy 44
  - Głuchy telefon 46
  - Trudno dojść do porozumienia 47
  - Lecz nie jest to niemożliwe 47
- (Nowe) podejście do strategii oprogramowania 49
  - Myślenie 49
  - Przemyślenie na nowo 51
- Czy monolity są złe? 54
- Czy mikrousługi są dobre? 55

Kup książkę Poleć książkę

Nie obwiniaj Agile ... 57
Wyrwać się z błota ... 59
Podsumowanie ... 60
Źródła ... 61

Rozdział 2. Podstawowe narzędzia strategicznego uczenia się ... 62
Decyzje: właściwe i niewłaściwe, wczesne i późne ... 63
Kultura i zespoły ... 66
Porażka to nie koniec ... 67
Kultura porażki to nie kultura zrzucania winy ... 68
Jak właściwie rozumieć prawo Conwaya? ... 69
Umożliwianie bezpiecznego eksperymentowania ... 73
Najpierw moduły ... 73
Wdrożenie na koniec ... 77
Wszystko pomiędzy ... 79
Zdolności biznesowe, procesy biznesowe i cele strategiczne ... 79
Celowe dostarczanie ... 83
Podejmowanie decyzji za pomocą Cynefin ... 86
Gdzie jest spaghetti i jak długo się je gotuje? ... 90
Architektura strategiczna ... 90
Zastosowanie narzędzi ... 92
Podsumowanie ... 95
Źródła ... 96

Rozdział 3. Eksperymentowanie i odkrywanie zorientowane na zdarzenia ... 98
Polecenia i zdarzenia ... 99
Stosowanie modeli oprogramowania ... 101
Szybkie uczenie się przy użyciu EventStormingu ... 102
Kiedy konieczne są sesje zdalne ... 104
Prowadzenie sesji ... 105
Modelowanie ogólnej wizji ... 109
Zastosowanie narzędzi ... 111
Podsumowanie ... 117
Źródła ... 118

Część II. Wspieranie innowacji biznesowych ... 119

Streszczenie ... 121

Rozdział 4. Ukierunkowanie na dziedzinę ... 125
Dziedziny i poddziedziny ... 127
Podsumowanie ... 130
Źródła ... 131

Rozdział 5. Wiedza kontekstowa 132
Kontekst ograniczony i język wszechobecny 132
Dziedzina główna 136
Poddziedziny pomocnicze, generyczne i mechanizmy techniczne 138
Poddziedziny pomocnicze 138
Poddziedziny generyczne 139
Mechanizmy techniczne 139
Zdolności biznesowe i konteksty 140
Nie za dużo, nie za mało 142
Podsumowanie 143
Źródła 144

Rozdział 6. Mapowanie, porażka i sukces — wybierz dwa 145
Mapowanie kontekstów 145
Partnerstwo 147
Wspólny rdzeń 149
Klient – Dostawca 150
Konformizm 152
Warstwa przeciwuszkodzeniowa 154
Usługa open-host 155
Język opublikowany 160
Osobne drogi 163
Modelowanie topografii 163
Ponoszenie porażek i odnoszenie sukcesów 166
Zastosowanie narzędzi 169
Podsumowanie 173
Źródła 174

Rozdział 7. Modelowanie konceptów dziedzinowych 175
Encje 176
Obiekty wartości 176
Agregaty 178
Usługi dziedzinowe 179
Zachowania funkcyjne 179
Zastosowanie narzędzi 181
Podsumowanie 182
Źródła 182

Część III. Architektura zorientowana na zdarzenia 183

Streszczenie 185

Rozdział 8. Architektura podstaw 189
Style architektoniczne, wzorce i czynniki decyzyjne 191
Porty i adaptery (architektura heksagonalna) 191
Modularyzacja 197
Zapytania/odpowiedzi REST 201

Kup ksi k Pole ksi k

Atrybuty jakości 203
Bezpieczeństwo 203
Prywatność 206
Wydajność 208
Skalowalność 210
Wytrzymałość — niezawodność i odporność na błędy 211
Złożoność 212
Zastosowanie narzędzi 213
Podsumowanie 213
Źródła 214

Rozdział 9. Architektury oparte na komunikatach i zdarzeniach 216
REST oparty na komunikatach i zdarzeniach 220
Dzienniki zdarzeń 221
Subscriber polling 222
Server-Sent Events 223
Zarządzanie procesami i oparte na zdarzeniach 224
Event Sourcing 227
CQRS 231
Serverless i Function as a Service 232
Zastosowanie narzędzi 234
Podsumowanie 234
Źródła 235

Część IV. Tworzenie przemyślanej architektury — dwie ścieżki 237

Streszczenie 239

Rozdział 10. Monolity na poważnie 243
Zarys historyczny 245
Poprawnie od samego początku 248
Zdolności biznesowe 248
Decyzje architektoniczne 251
Od chaosu do ładu 256
Zmiany na zmianach 258
Rozerwanie sprzężenia 261
Utrzymanie stanu właściwego 265
Podsumowanie 266
Źródła 267

Rozdział 11. Od monolitu do mikrousług 268
Przygotowanie mentalne 268
Od modularnego monolitu do mikrousług 271
Od monolitu wielkiej kuli błota do mikrousług 275
Interakcje użytkowników 276
Harmonizacja zmian danych 278
Co należy udusić? 283

Kup ksi k Pole ksi k

Odłączanie starszego monolitu 285
Podsumowanie 286
Źródła 287

Rozdział 12. Równowaga i strategia 288
Równowaga a atrybuty jakości 288
Strategia i cel 289
Cele biznesowe kierują transformacją cyfrową 289
Używanie narzędzi strategicznego uczenia się 290
Lekkie modelowanie oparte na zdarzeniach 291
Wspieranie innowacji biznesowych 292
Architektura zorientowana na zdarzenia 292
Monolity jako najważniejsze zagadnienie 293
Tworzenie mikrousług z monolitu 293
Równowaga wymaga bezstronności, innowacja jest niezbędna 294
Podsumowanie 295
Źródła 296

Kup ksi k Pole ksi k

Rozdział 1

Cele biznesowe i transformacja cyfrowa

Najznakomitszym osiągnięciem biznesowym jest stworzenie produktu, który spełnia zapotrzebowanie wielu konsumentów, jest całkowicie unikatowy i ma optymalną cenę. Historycznie i w sensie ogólnym dokonanie takiego osiągnięcia uzależnione było od umiejętności wskazania tego, co jest nieodzowne lub wysoce pożądane dla kluczowej grupy demograficznej na rynku. Znajduje to odzwierciedlenie w słowach Platona o państwie: „A będzie je budowała, jak się zdaje, nasza potrzeba". Dziś tę konstatację wyraża się powszechniej słowami „potrzeba matką wynalazków".

Największymi innowatorami są jednak ci, którzy potrafią wymyślić genialny produkt, zanim jeszcze konsumenci zdadzą sobie sprawę, że go potrzebują. Takie wyczyny zdarzały się przypadkowo, ale też w wyniku działania tych, którzy mieli odwagę spytać: „Czemu nie?"[1]. Być może matematyk i filozof Alfred North Whitehead trafił w sedno, twierdząc, że „podstawą wynalazku jest nauka, a nauka niemal w całości wyrasta z dającej rozkosz ciekawości intelektualnej" [ANW].

Zdecydowana większość przedsiębiorstw musi zmierzyć się ze smutną prawdą: przełomy w rozwoju produktów, które prowadzą do daleko idących zmian na rynku, nie zdarzają się na co dzień. Wynalezienie całkowicie wyjątkowych produktów, które opanują całe rynki, może wydawać się równie prawdopodobne, jak trafienie strzałą w sam środek tarczy z przewiązanymi oczami.

Dominującym planem biznesowym jest zatem stworzenie konkurencji dla innych. Wyjątkowość przejawia się tu raczej w określeniu ceny repliki niż w stworzeniu czegoś oryginalnego. Trafienie w tak duży cel jest czymś całkowicie zwyczajnym i nie wymaga wyobraźni, przy czym sukces wcale nie jest pewny. Jeśli nakręcanie konkurencji wydaje się najlepszym zagraniem, rozważ radę Steve'a Jobsa: „Nie możesz spojrzeć na konkurencję i stwierdzić, że zrobisz to samo, tylko lepiej. Musisz stwierdzić, że zrobisz to inaczej".

[1] (George) Bernard Shaw: „Niektórzy ludzie postrzegają rzeczy w ich naturalnym stanie i zadają sobie pytanie: dlaczego? Ja marzę o rzeczach, których nigdy nie było, i mówię sobie: dlaczego nie?".

Kup ksi k   Pole ksi k

Innowacyjność SpaceX

Pomiędzy rokiem 1970 a 2000 koszt wystrzelenia jednego kilograma ładunku w kosmos wynosił średnio 18,5 tysiąca dolarów. W przypadku SpaceX Falcon 9 koszt wynosi zaledwie 2720 dolarów za kilogram. To różnica rzędu 7:1, więc nie dziwota, że SpaceX obecnie panuje nad niemal całym rynkiem lotów w kosmos. Jak do tego doszło? SpaceX nie pracowało w ramach kontraktu rządowego, co do niedawna było jedynym mechanizmem finansowania takich przedsięwzięć. Celem firmy było drastyczne obniżenie kosztu wystrzeliwania ładunków w kosmos. Głównym celem pobocznym było z kolei odzyskiwanie i ponowne wykorzystywanie rakiet wspomagających. Na YouTubie można znaleźć cudowny film pokazujący wszystkie rakiety, które roztrzaskali, żeby ten cel ostatecznie osiągnąć. Strategia integrowania zdarzeń (czyli w tym przypadku testowych startów rakiet) pozwoliła różnym zespołom inżynieryjnym na szybkie, wspólne testowanie najnowszych wersji sprzętu. Na kontrakcie rządowym tak znaczna liczba niepowodzeń byłaby niedopuszczalna. Tymczasem to właśnie kraksy przyspieszyły rozwój niezawodnej, taniej rakiety, być może nawet pięciokrotnie. Wystarczyło wypróbowywać różne rozwiązania, aby poznać niewiadome niewiadome, zamiast starać się wszystko szczegółowo przemyśleć z wyprzedzeniem. To dość klasyczne podejście inżynieryjne, ale w modelu kontraktowym nigdy by się nie zgodzono na jego przyjęcie. Zespół SpaceX twierdzi, że wykonywanie nieudanych testów i odkrywanie problemów na bieżąco wychodziło znacznie taniej niż odwlekanie testów w nieskończoność, aby uniknąć ryzyka. [Mary Poppendieck]

Imitacja nie jest strategią. Jest nią dyferencjacja.

Dyferencjacja jest strategicznym celem biznesowym, do którego należy nieustannie dążyć. O ile trwanie w stanie czystej inwencji wydaje się niemal niemożliwe, nie można tego powiedzieć o stałym i wytrwałym doskonaleniu się w kierunku innowacji. W tej książce staramy się pomóc czytelnikom w osiągnięciu strategicznego zróżnicowania biznesowego poprzez ciągłe doskonalenie w zakresie transformacji cyfrowej.

Transformacja cyfrowa — co jest jej celem?

Zrozumienie, że tworzenie rzeczy niezwykłych jest niemałym dokonaniem, nie powinno nikogo odstręczać od wytrwałego stawiania małych, rozważnych kroków w kierunku rzeczywistej innowacji. Niezależnie od tego, jak dojście do punktu Z może być złożone, przeprowadzenie eksperymentu w celu dojścia do punktu B, kiedy zaczyna się od punktu A, wydaje się realistycznym podejściem. Można następnie przejść do punktu C, który prowadzi z kolei do punktu D. To kwestia tego, aby nie zdejmować fartucha laboratoryjnego i zaakceptować to, że wyjątkowe produkty zdolne do zdobywania nowych rynków prawdopodobnie znajdują się tuż przed naszym nosem.

Niezależnie od tego, czy pakiet Microsoft Office był uznawany już od początku za innowację w dziedzinie produktywności pracowników, z pewnością odniósł największy sukces

Kup ksi k Pole ksi k

na tym rynku. Kiedy Microsoft wprowadzał pakiet Office 365, nie musiał wymyślać edytora tekstu i arkusza kalkulacyjnego na nowo, aby wprowadzić innowację, lecz dodał nowy mechanizm dostawy i nowe funkcje, w tym ułatwiające pracę zespołową. Czy Microsoft po raz kolejny odniósł sukces, wprowadzając innowacje poprzez transformację cyfrową?

Transformacja cyfrowa pozostaje w gestii innowatora biznesowego, ale firmy często tracą z oczu jej innowacyjny aspekt. Dla transformacyjnej innowacji konieczne jest, aby firma zrozumiała różnicę między zmianą platform infrastrukturalnych a kreowaniem wartości produktu. Choć na przykład przeniesienie cyfrowych zasobów przedsiębiorstwa z centrum danych w siedzibie firmy do chmury może być ważnym przedsięwzięciem informatycznym, nie jest to samo w sobie innowacyjną inicjatywą biznesową.

Czy migrację oprogramowania do chmury można uznać za transformację cyfrową? Owszem, ale raczej wtedy, kiedy ma to służyć przyszłej dyferencjacji. Najlepiej, jeśli chmura zapewnia nowe możliwości wprowadzania innowacji lub przynajmniej obniżenia niezwykle wysokich kosztów eksploatacji zasobów cyfrowych i zainwestowania oszczędzonych środków w nowe produkty. Można powiedzieć, że w takim przypadku chmura tworzy nowe możliwości poprzez uwolnienie Cię od większości tradycyjnych obowiązków związanych z utrzymywaniem centrum danych. Nie mamy jednak do czynienia z transformacją, jeśli przejście do chmury oznacza zamianę jednego zestawu kosztów na inny. Dla Amazona zaoferowanie swojej dobrze działającej infrastruktury obliczeniowej zewnętrznym podmiotom było transformacją cyfrową, której rezultatem była innowacja w zakresie chmury. Płacenie abonamentu Amazonowi za korzystanie z jego chmury nie jest transformacyjną innowacją dla klienta. Morał z tego jest jasny: innowacje albo wprowadzasz sam, albo ktoś je wprowadza na Tobie.

Tak samo jak migracja do chmury nie jest innowacją, nie jest nią też tworzenie nowej architektury obliczeń rozproszonych. Użytkowników nie obchodzą rozwiązania rozproszone, mikrousługi czy monolity, a nawet cechy produktu. Obchodzą ich wyniki. Konieczne jest zapewnienie użytkownikom lepszych wyników szybko i bez niekorzystnego wpływania na ich obieg pracy. Aby oprogramowanie miało szansę na dokonanie znaczącej transformacji, jego architektura i projekt muszą sprzyjać dostarczaniu użytkownikom lepszych wyników tak szybko, jak to możliwe.

W przypadku korzystania z chmury ulepszone podejście do architektury i projektu (oraz wszelkie dodatkowe, dopracowane działania prowadzące do zwiększenia wydajności) umożliwia osiągnięcie innowacyjnych celów transformacji. Korzystanie z infrastruktury jako usługi daje firmie przestrzeń do pracy nad innowacyjnym oprogramowaniem biznesowym, ponieważ nie musi ona starać się wprowadzać innowacji w infrastrukturze. Innowacje infrastrukturalne są nie tylko czasochłonne i kosztowne, ale mogą też nie przynosić firmie korzyści finansowych. Infrastruktura tworzona we własnym zakresie może ponadto nie zaspokajać potrzeb infrastrukturalnych i operacyjnych tak dobrze, jak AWS, Google Cloud Platform i Azure. Nie zawsze tak jednak musi być. W niektórych przypadkach znacznie bardziej opłacalne jest przeniesienie działalności operacyjnej do firmy lub utrzymanie jej tam [a16z-CloudCostParadox].

Pamiętaj: ścieżka przebiega od A do B, od B do C, od C do D… Bądź gotów do wielokrotnego powtarzania tych kroków, aby zebrać wiedzę pozwalającą na podjęcie kolejnych. Świadomość tego, że można oczekiwać cofnięcia się z J do G przed dotarciem do K oraz że dotarcie do punktu Z wcale nie musi być konieczne, jest wyzwalająca. Zespoły mogą wymyślać innowacje, ale żaden z tych etapów transformacji nie toleruje długich cykli. W rozdziale 2. „Podstawowe

Kup ksi k Pole ksi k

narzędzia strategicznego uczenia się" przekonasz się, że eksperymentowanie sprzyja innowacji i pozwala oprzeć się niedecyzyjności.

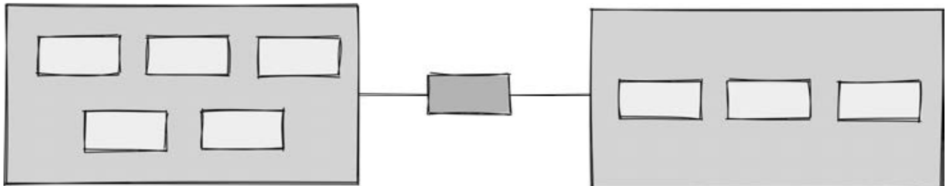
Architektura oprogramowania — szybki przegląd

Ten punkt dotyczy pojęcia **architektura oprogramowania**, które często jest w tej książce przywoływane. To dość szeroki temat, który zostanie omówiony bardziej szczegółowo w dalszej części książki.

Na razie o architekturze oprogramowania możesz myśleć jak o architekturze budynku. Budynek ma pewną strukturę, która jest odzwierciedleniem rezultatów komunikacji pomiędzy architektem a właścicielem na temat cech projektowych i zbiorem cech, które zostały w wyniku tego opracowane. Budynek stanowi całościowy system składający się z różnych podsystemów, z których każdy ma swój własny cel i rolę. Wszystkie te podsystemy są luźniej lub ściślej powiązane z innymi częściami budynku, pracując oddzielnie lub w połączeniu z innymi, tak aby budynek spełniał swoje zadanie. Na przykład klimatyzacja budynku wymaga zasilania elektrycznego, kanałów wentylacyjnych, termostatu, izolacji, a nawet wydzielonej części budynku, która ma być chłodzona, żeby ten podsystem miał sprawnie działać.

Architektura oprogramowania, analogicznie, określa projekt strukturalny, czyli postać wielu struktur, a nie jednej. Projekt strukturalny organizuje części składowe systemu, zapewniając im możliwość komunikacji podczas wspólnej pracy. Struktura także rozdziela grupy komponentów, aby mogły one funkcjonować niezależnie od siebie. Struktury muszą zatem pomagać w osiąganiu cech jakości, a nie funkcjonalności, podczas gdy zawarte w nich komponenty odpowiadają za funkcjonalność określoną przez zespoły twórców systemu.

Rysunek 1.1 przedstawia dwa podsystemy (i tym samym ukazuje zaledwie fragment całego systemu), przy czym oba mają komponenty współpracujące ze sobą wewnętrznie, ale w odizolowaniu od drugiego podsystemu. Oba podsystemy wymieniają informacje za pośrednictwem kanału komunikacyjnego. Znajdujący się pomiędzy nimi blok reprezentuje wymieniane informacje. Załóżmy, że te dwa podsystemy są fizycznie rozdzielone na dwie jednostki wdrożeniowe i komunikują się przez sieć. Stanowią one wówczas część systemu rozproszonego.



Rysunek 1.1. *Architektura oprogramowania określa strukturę podsystemów i wspiera komunikację między nimi*

Innym istotnym aspektem architektury budynków i oprogramowania jest to, że musi być przygotowana na nieuchronne zmiany. Jeśli istniejące komponenty nie spełnią nowych wymagań w którejkolwiek z tych architektur, zastąpienie ich bez ponoszenia ogromnych kosztów i dużego wysiłku musi być możliwe. Architektura musi też umożliwiać ewentualną konieczną rozbudowę w sposób, który nie wpływa na nią samą.

Kup ksi k Pole ksi k

Dlaczego oprogramowanie się nie sprawdza?

Nie chcemy wyolbrzymiać złego stanu dziedziny rozwoju oprogramowania dla przedsiębiorstw, ale nie sądzimy też, żeby można go było wyolbrzymić.

Z dyskusji o stanie systemów oprogramowania dla przedsiębiorstw z firmami z listy „Fortune" i „Global" szybko dowiadujemy się o ich głównych bolączkach. Są one zawsze związane ze starzejącym się oprogramowaniem, które jest utrzymywane i serwisowane przez dziesięciolecia, długo po wprowadzeniu samej innowacji. W większości omówień wskazuje się na to, że rozwój oprogramowania jest traktowany jako miejsce generowania kosztów przedsiębiorstwa, co znacznie utrudnia inwestowanie w ulepszenia. Oprogramowanie współcześnie powinno stanowić jednak centrum generowania zysków. Niestety, zbiorowa mentalność korporacyjna zatrzymała się w rozwoju ponad 30 lat temu, kiedy oprogramowanie miało sprawić, by niektóre zadania były wykonywane szybciej, niż mógłby to robić człowiek ręcznie.

Tworzenie aplikacji (lub podsystemu) zaczyna się od zasadniczego celu biznesowego. Z biegiem czasu jej podstawowy cel ulega rozszerzeniu lub może nawet zostać istotnie zmieniony. Ciągłe dodawanie funkcji może poszerzyć jej funkcjonalność w takim stopniu, że jej pierwotny cel zostaje zapomniany, przez co aplikacja może być różnie rozumiana w kontekście różnych funkcji biznesowych, przy czym trudno ogarnąć rozumem pełną różnorodność tych interpretacji. Często prowadzi to do sytuacji, „gdzie kucharek sześć". Ostatecznie rozwój strategiczny ustępuje utrzymywaniu oprogramowania w działaniu poprzez naprawianie pojawiających się nagle błędów i łatanie danych bezpośrednio w bazie danych w ramach przeciwdziałania usterkom. Nowe funkcje są zazwyczaj dodawane powoli i ostrożnie, aby uniknąć wystąpienia jeszcze większej liczby błędów. Wprowadzanie nowych błędów jest mimo to nieuniknione. Przy stale rosnącym poziomie nieuporządkowania systemu i utracie pierwotnej wizji niemożliwe jest określenie pełnego wpływu, jaki określona zmiana będzie miała na całość oprogramowania.

Zespoły przyznają, że nie mają jasnego i intencjonalnego sposobu formułowania architektury oprogramowania, w tym ani wobec poszczególnych aplikacji (podsystemów), ani nawet ogólnie w odniesieniu do dowolnego dużego systemu. Tam, gdzie istnieje jakakolwiek architektura, jest ona zazwyczaj krucha i przestarzała, zważywszy na postępy, jakie dokonały się w projektowaniu sprzętu i środowisk operacyjnych takich jak chmura. Projektowanie oprogramowania również odbywa się mimochodem, a przez to wydaje się, jakby projekt w ogóle nie istniał. W wyniku tego większość założeń przyświecających implementacji jest ukryta i tkwi wyłącznie w pamięci kilku osób, które nad nią pracowały. Zarówno architektura, jak i projekt są przeważnie tworzone doraźnie i wyglądają po prostu dziwnie. Te nierozpoznane błędy sprawiają, że w wyniku niedbałej pracy uzyskuje się naprawdę niechlujne wyniki.

Równie niebezpieczne jak brak dobrze zdefiniowanej architektury jest wprowadzanie architektury z przyczyn czysto technicznych. Wśród architektów oprogramowania i programistów często pojawia się fascynacja nowym stylem tworzenia oprogramowania względem tego, który stosowali wcześniej, a nawet nowym narzędziem, wokół którego słychać dużo szumu w branży. Na ogół wprowadza to przypadkową złożoność[2], ponieważ pracownicy IT nie do

[2] Przyczyną przypadkowej złożoności są próby, jakie programiści podejmują w celu rozwiązywania problemów. Można ją jednak skorygować. Niektóre rodzaje oprogramowania odznaczają się też niezbędną

Kup ksi k Pole ksi k

końca rozumieją wpływ, jaki ich nierozważne decyzje wywrą na system jako całość, w tym na jego środowisko wykonawcze i działanie. Architektura mikrousług i narzędzia takie jak Kubernetes, choć mają odpowiednie zastosowanie w odpowiednim kontekście, napędzają nieuzasadnione wprowadzanie nowinek do użytku. Niestety, rzadko wynika to z realnego wglądu w potrzeby biznesowe.

Długotrwałe narastanie nieścisłości modelu oprogramowania w systemie, wynikające z niezdolności do wprowadzenia pilnych zmian, opisuje się **metaforą długu**. Z kolei nagromadzenie niekontrolowanych zmian w systemie jest znane jako **entropia oprogramowania**. Obu tym zjawiskom warto przyjrzeć się bliżej.

Metafora długu

Dziesiątki lat temu Ward Cunningham, niezwykle pojętny programista, który pracował wówczas nad oprogramowaniem finansowym, musiał wyjaśnić swojemu przełożonemu, dlaczego działania na rzeczy modyfikacji programowania były konieczne [Cunningham]. Wprowadzane zmiany nie były doraźne — wręcz przeciwnie. Wyglądało to tak, jakby programiści doskonale wiedzieli, co robią, i można było odnieść wrażenie, że zrobienie tego jest proste. Technika, z której wówczas skorzystali, znana jest obecnie jako **refaktoryzacja oprogramowania**. W tym przypadku refaktoryzacja została przeprowadzona w zamierzony sposób, czyli tak, aby odzwierciedlić w modelu oprogramowania nowo nabytą wiedzę biznesową.

Aby uzasadnić tę pracę, Cunningham musiał wyjaśnić, że jeśli zespół nie wprowadzi do oprogramowania zmian dopasowujących je do coraz lepszego zrozumienia dziedziny problemu, to ciągle się borykał z niezgodnością dotychczasowego programowania z aktualną, poszerzoną wiedzą. To z kolei spowolniłoby postępy zespołu w ciągłym rozwoju, co przypominałoby *spłacanie odsetek od kredytu*. W taki oto sposób narodziła się **metafora długu**.

Każdy może pożyczyć pieniądze, aby umożliwić sobie wykonanie pewnych czynności wcześniej, niż gdyby tych pieniędzy nie miał. Rzecz jasna, dopóki dług jest zaciągnięty, pożyczkobiorca musi płacić odsetki. Głównym założeniem zaciągania długu w dziedzinie wytwarzania oprogramowania jest umożliwienie szybszego wydania produktu, ale ze świadomością tego, że lepiej go spłacić wcześniej niż później. Odbywa się to poprzez refaktoryzację oprogramowania w celu odzwierciedlenia nowo nabytej wiedzy zespołu o potrzebach biznesowych. W tamtych czasach, podobnie jak dzisiaj, oprogramowanie dostarczano użytkownikom w pośpiechu, na ogół ze świadomością istnienia owego długu. Zespoły jednak aż nadto często trwały w przekonaniu, że spłata nigdy nie będzie konieczna.

Doskonale jednak wiemy, co się w takich przypadkach dzieje. Kiedy dług narasta, a dana osoba pożycza coraz więcej, wszystkie środki pożyczkobiorcy idą na spłatę odsetek, a on sam dochodzi do punktu, w którym traci jakąkolwiek siłę nabywczą. Podobnie jest z zadłużeniem oprogramowania — programiści pogrążeni w długach ostatecznie uświadamiają sobie, że znajdują się w bardzo trudnej sytuacji. Dodawanie nowych funkcji trwa coraz dłużej, a w końcu dochodzi do momentu, w którym zespół nie jest w stanie uzyskać prawie żadnych postępów.

złożonością, wynikającą z charakteru rozwiązywanych problemów. Choć niezbędnej złożoności nie można uniknąć, można ją wyizolować w podsystemach i komponentach zaprojektowanych konkretnie z myślą o jej obsłudze.

Kup ksi k Pole ksi k

Jednym z głównych problemów ze współczesnym rozumieniem metafory długu jest to, że wielu programistów uważa, że sprzyja ona celowemu dostarczaniu źle zaprojektowanego i wdrożonego oprogramowania, aby szybciej dostarczyć produkt. Tymczasem wcale tak nie jest. Próba dokonania czegoś takiego przypomina raczej zaciąganie kredytów subprime[3] o stopie procentowej zmiennej w górę, co często prowadzi do nadmiernego finansowego obciążenia kredytobiorcy, aż ten przestaje być zdolny do wywiązania się ze zobowiązań. Dług przydaje się tylko wtedy, kiedy jest pod kontrolą; w przeciwnym razie wywołuje niestabilność w obrębie całego systemu.

Entropia oprogramowania

Entropia oprogramowania[4] jest odmienną metaforą, ale powiązaną z pojęciem długu pod względem opisywanego przez nią stanu systemu oprogramowania. Słowa **entropia** używa się w mechanice statystycznej w dziedzinie termodynamiki do pomiaru nieuporządkowania systemu. Nie próbując zbytnio zagłębiać się w ten temat, można powiedzieć, że drugie prawo termodynamiki głosi, iż *entropia izolowanego systemu nigdy nie maleje z czasem*. Izolowane systemy spontanicznie ewoluują w kierunku równowagi termodynamicznej, stanu o maksymalnej entropii" [Entropia]. Metafora entropii oprogramowania określa stan systemu oprogramowania, w którym zmiana jest nieunikniona, i wskazuje, że owa zmiana spowoduje wzrost niekontrolowanej złożoności, o ile nie podejmie się energicznych wysiłków, aby temu zapobiec [Jacobson].

Wielka kula błota

Aplikację lub system taki jak opisany wcześniej nazywa się **wielką kulą błota**. Pod względem architektonicznym opisuje się taki system jako bezładny, rozległy, niechlujny, sklejony na ślinę, zarośnięty, rozrastający się niekontrolowanie i wymagający powtarzalnych, doraźnych napraw. „Informacje są bezładnie przekazywane między odległymi elementami systemu, często w takim stopniu, że niemal wszystkie ważne informacje stają się globalne lub powielone. Możliwe, że ogólna struktura systemu nigdy nie była zdefiniowana, a jeśli była, to uległa erozji i jest nie do rozpoznania" [BBoM].

Celne wydaje się określenie „architektury" wielkiej kuli błota mianem *nie*architektury.

W dalszej części tego rozdziału, a także w całej książce, będziemy zwracać uwagę na niektóre z wymienionych powyżej cech: bezładną strukturę, niekontrolowany rozrost, powtarzalne i doraźne naprawy, bezładne przekazywanie informacji, globalny lub powielony status wszystkich ważnych informacji.

---

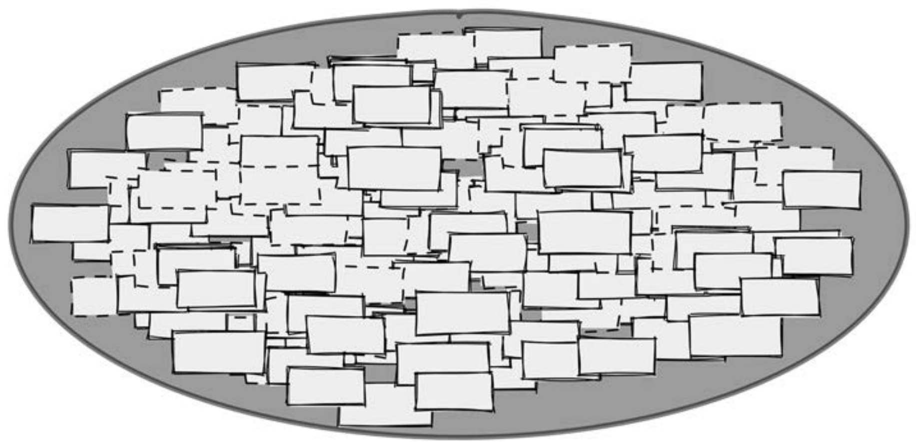
[3] Trudno pojąć, że niektórzy jakby nie dostrzegli kryzysu finansowego z 2008 roku, który trwał latami. Ten (ostatecznie globalny) kryzys został wywołany przez udzielanie kredytów subprime niekwalifikującym się kredytobiorcom na zakup nieruchomości. Niektórzy pierwsi czytelnicy szkicu tej książki pytali o to, czym taki kredyt właściwie jest. Zapoznanie się z tą historią może pomóc uchronić czytelników od takich finansowych problemów.

[4] Poza entropią istnieją także inne analogie, które równie żywo obrazują problem: gnicie, erozja i rozpad oprogramowania. Autorzy głównie jednak używają pojęcia entropii.

Kup ksi k Pole ksi k

Traktowanie wielkiej kuli błota jako normy powoduje, że organizacje doświadczają paraliżu konkurencyjnego, co rozprzestrzeniło się już na wszystkie branże. Nierzadko bywa tak, że duże przedsiębiorstwa, które niegdyś wyróżniały się na tle konkurencji, są spętane przez systemy o głębokim zadłużeniu i niemal całkowitej entropii.

System wielkiej kuli błota przedstawiony na rysunku 1.2 można łatwo skontrastować z tym z rysunku 1.1. Oczywiście, segment systemu z rysunku 1.1 nie odzwierciedla łącznej liczby funkcji obsługiwanych przez system z rysunku 1.2, ale wyraźnie widać, że architektura pierwszego systemu ustanawia porządek, podczas gdy jej brak w drugim wywołuje chaos.



Rysunek 1.2. *Wielka kula błota może być postrzegana jako niearchitektura*

Takie chaotyczne warunki uniemożliwiają wydanie więcej niż kilku wersji oprogramowania rocznie, a te z kolei powodują jeszcze gorsze problemy od tych, które wydawane były w minionych latach. Jednostki i zespoły, do których należą, mają skłonność do obojętnienia i spoczywania na laurach, ponieważ wiedzą, że nie są w stanie wprowadzić zmian, które są ich zdaniem konieczne, aby zmienić ten stan rzeczy. Kolejnym etapem jest rozczarowanie i demoralizacja. Firmy znajdujące się w takiej sytuacji nie są zdolne do wprowadzania innowacji w oprogramowaniu i dalszego konkurowania. Ostatecznie padają ofiarą rzutkiego start-upu, który jest zdolny do sprawnego działania i wyparcia z rynku dotychczasowych liderów w ciągu paru miesięcy lub lat.

Bieżący przykład

Od tego miejsca będziemy omawiać studium przypadku oparte na wielkiej kuli błota i opisywać sytuację, w której dotknięte nią przedsiębiorstwo walczy o własną innowacyjność, stawiając czoła przeciwnościom związanym z głębokim zadłużeniem i entropią. Ponieważ możesz być już zmęczony samymi złymi wiadomościami, możemy Cię pocieszyć: z czasem sytuacja ulega poprawie.

Nie ma lepszego sposobu na wyjaśnienie problemów, z jakimi musi się zmierzyć każda firma zajmująca się wytwarzaniem oprogramowania, od sięgnięcia po przykłady rzeczywistego

Kup ksi k Pole ksi k

świata. Przedstawiony tutaj przykład, będący studium przypadku radzenia sobie z istniejącą wielką kulą błota, pochodzi z branży ubezpieczeniowej.

Na pewnym etapie życia niemal każdy ma do czynienia z firmą ubezpieczeniową. Istnieje wiele powodów, dla których ludzie starają się o różne polisy ubezpieczeniowe. Czasami chodzi o spełnienie wymogów prawnych, a w innych przypadkach o zapewnienie sobie bezpieczeństwa na przyszłość. Polisy te obejmują ochronę zdrowia, ubezpieczenia osobiste, w tym na życie, samochodowe, mieszkaniowe, hipoteczne, inwestycje w produkty finansowe, podróże międzynarodowe, a nawet utratę ulubionego zestawu kijów golfowych. Innowacyjność produktów ubezpieczeniowych w dziedzinie ubezpieczeń wydaje się być nieskończona, gdyż można nimi objąć niemal każde wyobrażalne ryzyko. Jeśli istnieje potencjalne ryzyko, prawdopodobnie znajdzie się firma ubezpieczeniowa, która zapewni jego pokrycie.

Sama koncepcja ubezpieczeń polega na tym, że jakaś osoba lub rzecz jest narażona na ryzyko straty i, za opłatą, możliwe jest odzyskanie obliczonej wartości finansowej ubezpieczonej osoby lub rzeczy, jeżeli taka strata nastąpi. Ubezpieczenie jest udaną propozycją biznesową ze względu na prawo wielkich liczb. Mówi ono, że przy wielkiej liczbie osób i rzeczy objętych ubezpieczeniem od ryzyka, całkowite ryzyko straty wśród nich wszystkich jest dość małe, wobec czego suma wnoszonych opłat jest znacznie wyższa od sumy wypłat z tytułu poniesionych strat. Ponadto im większe jest prawdopodobieństwo wystąpienia szkody, tym wyższa jest opłata, jaką towarzystwo ubezpieczeniowe pobiera za ochronę.

Wyobraźmy sobie złożoność dziedziny ubezpieczeń. Czy ochrona ubezpieczeniowa pojazdów i nieruchomości jest jednakowa? Czy zmodyfikowanie kilku reguł biznesowych, które mają zastosowanie do pojazdów, umożliwia objęcie ubezpieczeniem nieruchomości? Nawet jeśli polisy samochodowe i nieruchomościowe można uznać za „wystarczająco podobne", pomyśl o odmiennych rodzajach ryzyka związanych z tymi dwoma rodzajami polis.

Rozważmy kilka przykładowych scenariuszy. Prawdopodobieństwo, że samochód uderzy w inny samochód, jest znacznie większe niż prawdopodobieństwo, że część domu uderzy w inny dom i spowoduje szkody. Prawdopodobieństwo wystąpienia pożaru w kuchni w wyniku normalnego codziennego użytkowania jest większe niż prawdopodobieństwo zapalenia się silnika samochodu w takich samych warunkach. Jak widać, różnica między tymi dwoma rodzajami ubezpieczeń nie jest drobna. Biorąc pod uwagę różnorodność możliwych zakresów ubezpieczenia, zaoferowanie polis, które będą wartościowe dla osób narażonych na ryzyko i nie przyniosą strat firmie ubezpieczeniowej, wymaga znacznego zaangażowania.

Zrozumiałe jest zatem, że złożoność strategii biznesowej, operacji i rozwoju oprogramowania w firmach ubezpieczeniowych jest duża. Z tego też względu firmy ubezpieczeniowe specjalizują się w niewielkim zakresie produktów, które podlegają ubezpieczeniu. Nie chodzi o to, że nie chciałyby objąć większego segmentu rynku, ale raczej o to, że koszty mogłyby przewyższyć korzyści wynikające z konkurowania we wszystkich możliwych segmentach. Nic więc dziwnego, że firmy ubezpieczeniowe częściej starają się być liderami w zakresie produktów ubezpieczeniowych, w których mają już doświadczenie. Niemniej dostosowanie strategii biznesowych, zaakceptowanie nieokreślonego, lecz wymiernego ryzyka oraz opracowanie nowych produktów może być zbyt lukratywną szansą, aby z niej nie skorzystać.

Nadszedł czas, aby przedstawić NuCoverage Insurance. Opis tej fikcyjnej firmy oparty jest na rzeczywistych scenariuszach, z którymi autorzy mieli wcześniej do czynienia. NuCoverage stało się liderem na rynku tanich ubezpieczeń samochodowych w Stanach Zjednoczonych.

Kup ksi k Pole ksi k

Firmę założono w 2001 roku z planem biznesowym zakładającym skupienie się na oferowaniu tańszych składek dla kierowców, którzy jeżdżą bezpiecznie. Dostrzegła ona szansę w skoncentrowaniu się na tym konkretnie rynku i rzeczywiście odniosła sukces, co wzięło się ze zdolności firmy do bardzo dokładnej oceny ryzyka i wyceniania składek oraz oferowania najtańszych polis na rynku. Blisko 20 lat później firma ubezpiecza 23% rynku amerykańskiego w ogóle, ale prawie 70% wyspecjalizowanego rynku tańszych ubezpieczeń dla jeżdżących bezpiecznie kierowców.

Aktualny kontekst biznesowy

Choć firma NuCoverage jest liderem w branży ubezpieczeń samochodowych, chciałaby rozszerzyć swoją działalność o inne rodzaje ubezpieczeń. Niedawno wprowadziła ubezpieczenia nieruchomości i pracuje nad ubezpieczeniami indywidualnymi. Dodanie nowych produktów ubezpieczeniowych okazało się jednak bardziej skomplikowane, niż początkowo zakładano.

W czasie kiedy rozwijano ubezpieczenia indywidualne, kierownictwo firmy miało okazję podpisać umowę partnerską z jednym z największych amerykańskich banków, WellBankiem. Umowa dotyczyła umożliwienia WellBankowi sprzedaży ubezpieczeń samochodowych pod własną marką. WellBank dostrzega wielki potencjał w sprzedaży ubezpieczeń samochodowych poza samym oferowaniem kredytów samochodowych. NuCoverage odpowiada zatem za polisy samochodowe WellBanku.

Oczywiście ubezpieczenia samochodowe NuCoverage różnią się od tych, które będzie sprzedawać WellBank. Najbardziej widoczne różnice związane są z poniższymi obszarami:

- składkami i zakresem ubezpieczenia,
- regulaminem i określaniem wysokości składki,
- oceną ryzyka.

Pomimo że NuCoverage nigdy dotąd nie brało udziału w takim partnerstwie, liderzy biznesowi natychmiast dostrzegli potencjał zwiększenia swojego zasięgu i być może nawet wprowadzenia zupełnie nowej i innowacyjnej strategii biznesowej. Jaką jednak formę miałaby przyjąć?

Okazja biznesowa

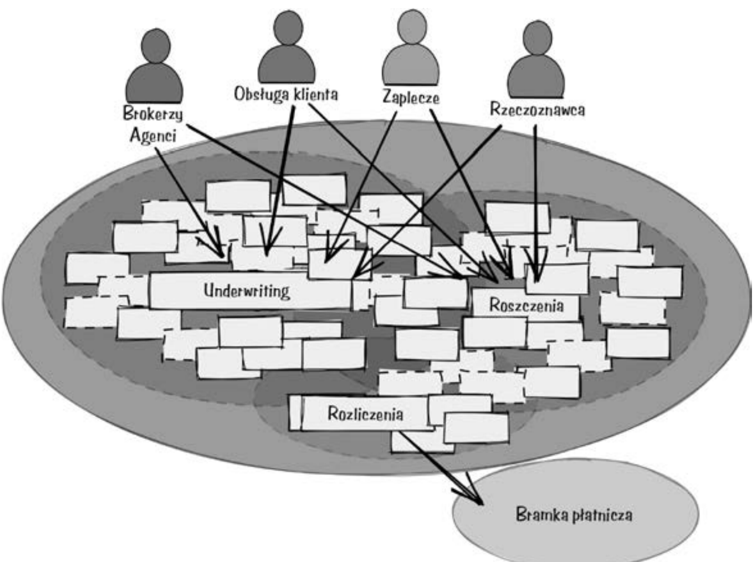
Zarząd i dyrektorzy NuCoverage dostrzegli jeszcze większą strategiczną szansę niż partnerstwo z WellBankiem: mogliby wprowadzić platformę ubezpieczeniową typu *white label*[5], która obsługiwałaby dowolną liczbę początkujących ubezpieczycieli. Wiele rodzajów działalności mogłoby potencjalnie sprzyjać sprzedaży produktów ubezpieczeniowych pod własną marką. Każda firma zna swoich klientów najlepiej i wie, jakie produkty ubezpieczeniowe można im zaoferować. Niedawno zawarte partnerstwo z WellBankiem jest tylko jednym z przykładów. NuCoverage może z pewnością wskazać innych myślących perspektywicznie partnerów, którzy podzielaliby wizję sprzedaży produktów ubezpieczeniowych pod marką typu white label.

---

[5] Produkt typu *white label* jest produktem lub usługą dostarczaną przez jedną firmę (producenta), który inne firmy (sprzedawcy) opatrują inną marką, aby wywołać wrażenie, że to one same go wytworzyły.

NuCoverage może na przykład nawiązać współpracę z producentami samochodów, którzy oferują własne finansowanie. Kiedy klient kupuje samochód, dealer może zaoferować zarówno finansowanie, jak i ubezpieczenie pod marką producenta. Możliwości jest multum, ponieważ nie każda przypadkowa firma może z łatwością stać się firmą ubezpieczeniową, ale za to wciąż może korzystać na marżach uzyskiwanych dzięki sprzedaży ubezpieczeń. W dłuższej perspektywie NuCoverage rozważa dywersyfikację poprzez wprowadzenie nowych produktów ubezpieczeniowych, takich jak ubezpieczenia motocykli, jachtów, a nawet zwierząt domowych.

Zarząd i kierownictwo entuzjastycznie podeszło do takiego prospektu, ale wieści o tych planach okazały się trudne do przełknięcia dla zespołu zarządzającego tworzeniem oprogramowania. Pierwotna aplikacja do obsługi ubezpieczeń samochodowych została opracowana naprędce i pod dużą presją, co szybko doprowadziło do powstania monolitu w postaci wielkiej kuli błota. Jak widać na rysunku 1.3, po ponad 20 latach wprowadzania zmian i niespłacania długu technicznego, a także w związku z ciągłym rozwojem systemu dla ubezpieczeń indywidualnych zespoły doszły do etapu, na którym nieplanowana złożoność ograniczała ich możliwości działania. Istniejące oprogramowanie absolutnie nie jest w stanie wspomagać osiągania aktualnych celów biznesowych. Pomimo tego dział rozwoju i tak musi zmierzyć się z tym wyzwaniem.



Rysunek 1.3. *Wielka kula błota NuCoverage. Wszystkie działania biznesowe przeplatają się z pogmatwanymi komponentami oprogramowania, które są obciążone znacznym długiem i których entropia zbliża się do maksimum*

NuCoverage musi zrozumieć, że działalność firmy nie ogranicza się już wyłącznie do ubezpieczeń. Zawsze była to firma produktowa, z tym że jej produktami były polisy ubezpieczeniowe. Jej transformacja cyfrowa sprawia, że firma staje się przedsiębiorstwem technologicznym, a jej produkty obejmują też oprogramowanie. Z tego względu NuCoverage musi

Kup ksi k Pole ksi k

zacząć myśleć jak firma technologiczna i podejmować decyzje, które wspierają takie pozycjonowanie — nie tylko doraźnie, lecz w perspektywie długoterminowej. To bardzo ważna zmiana w mentalności firmowej. Transformacja cyfrowa NuCoverage nie powiedzie się, jeśli będzie napędzana wyłącznie przez decyzje technologiczne. Kierownictwo firmy musi skoncentrować się na przekształceniu mentalności członków organizacji, a także kultury i procesów organizacyjnych, zanim zdecyduje, jakich narzędzi cyfrowych użyć i w jaki sposób.

Twoje przedsiębiorstwo a prawo Conwaya

Dawno temu (a konkretnie w 1967 roku) w całkiem niedalekiej galaktyce (zresztą naszej własnej) inny bardzo błyskotliwy twórca oprogramowania przedstawił pewien nieuchronny fakt o rozwoju systemów. Jest on do tego stopnia nie do uniknięcia, że stał się znany jako prawo. Programistą tym był Mel Conway, a fakt stał się znany jako prawo Conwaya.

> Prawo Conwaya: „Organizacje, które projektują systemy, ograniczone są do tworzenia projektów będących kopiami własnych struktur komunikacyjnych" [Conway].

Korelacja powyższego opisu z wielką kulą błota wydaje się dość oczywista. To w zasadzie niesprawna komunikacja odpowiada za „bezładną strukturę, niekontrolowany rozrost, powtarzalne i doraźne naprawy".

Ponadto niemal zawsze brakuje pewnego innego istotnego komponentu komunikacyjnego, jakim jest utrzymywanie produktywnej komunikacji pomiędzy interesariuszami biznesowymi a technicznymi, prowadzącej do głębokiego uczenia się, które z kolei prowadzi do innowacji.

> Twierdzenie: ci, którzy chcą tworzyć dobre, innowacyjne oprogramowanie, muszą w pierwszej kolejności poprawnie wyznaczyć ścieżkę na linii komunikacja – uczenie się – innowacja.

Z tymi prawami jest zabawnie. Czy możliwe jest „stanie się lepszym" z danego prawa? Człowiek na przykład nie może za bardzo „stać się lepszy" z prawa ciążenia. Wiemy, że po podskoczeniu opadniemy. Prawo ciążenia i znajomość oddziaływania grawitacyjnego ziemi pozwala nam nawet obliczyć, jak długo będziemy unosić się nad ziemią po wykonaniu skoku. Niektórzy ludzie potrafią skakać wyżej i dalej, ale wciąż podlegają temu samemu prawu ciążenia, co wszyscy inni na Ziemi.

Podobnie jak w przypadku prawa ciążenia, nie możemy w zasadzie „stać się lepszymi" z prawa Conwaya. Podlegamy mu. To prawo najlepiej sobie przyswoić, tak aby stać się lepszym w dziedzinie radzenia sobie z nieuchronnymi konsekwencjami jego obowiązywania. Rozważmy zatem wyzwania i możliwości.

Komunikacja dotyczy wiedzy

Wiedza jest najważniejszym zasobem każdej firmy. Organizacja nie może górować we wszystkim, więc musi wybrać swoje podstawowe kompetencje. Konkretna wiedza w zakresie danej dziedziny, jakiej przedsiębiorstwo nabywa, umożliwia uzyskanie przewagi konkurencyjnej.

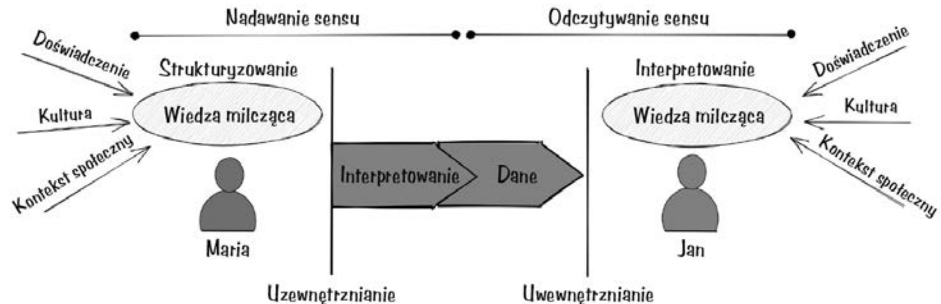
Kup ksi k Pole ksi k

Choć wiedza firmy może zmaterializować się w postaci fizycznych wytworów w rodzaju dokumentacji, a także w modelach i algorytmach powstających przy wdrażaniu kodu źródłowego, nie są one porównywalne ze zbiorową wiedzą pracowników. Wiedza jest w większości zawarta w indywidualnych umysłach. Wiedzę, która nie została uzewnętrzniona, określa się mianem **wiedzy milczącej**. Może ona należeć do zbiorowości, tak jak ma to miejsce w przypadku wykonywanych w obrębie organizacji rutynowych procedur, których przebieg nigdzie nie jest udokumentowany, albo i mieć zupełnie indywidualny charakter, tak jak w przypadku najbardziej odpowiadających poszczególnym pracownikom sposobów wykonywania swojej pracy. Wiedza osobista dotyczy umiejętności i fachowości — nieudokumentowanych tajemnic zawodowych, a także historycznej i kontekstowej wiedzy, którą przedsiębiorstwo gromadzi od początku działalności.

Członkowie organizacji wymieniają się wiedzą w toku skutecznej komunikacji. Im sprawniej odbywa się komunikacja, tym lepiej organizacja radzi sobie z dzieleniem się wiedzą. Nie odbywa się to jednak statycznie, tak jakby wczytywano w jednostki encyklopedyczne dane bez żadnych dodatkowych korzyści. Dzielenie się wiedzą z myślą o osiągnięciu danego celu prowadzi do uczenia się, a doświadczenie zbiorowego uczenia się może zaowocować przełomową innowacją.

Wiedza nie jest wytworem

Ponieważ wiedza nie jest czymś, co jedna osoba przekazuje drugiej w taki sam sposób jak fizyczny przedmiot, transfer wiedzy zachodzi jako połączenie **nadawania sensu** (ang. *sense-giving*) i **odczytywania sensu** (ang. *sense-reading*), jak widać na rysunku 1.4 [Polanyi].



Rysunek 1.4. *Przekazywanie wiedzy milczącej w toku nadawania i odczytywania sensu*

Nadawanie sensu ma miejsce, kiedy człowiek przekazuje wiedzę. Wiedza zostaje ustrukturyzowana i przyjmuje postać informacji, a następnie ulega uzewnętrznieniu [LAMSADE]. Odbiorca przechodzi proces odczytywania sensu. Ta jednostka wydobywa dane z otrzymanej informacji, tworząc tym samym osobistą wiedzę i uwewnętrzniając ją. Prawdopodobieństwo, że dwoje ludzi nada to samo znaczenie tej samej informacji, określone jest nie tylko dokładnością komunikacji zachodzącej pomiędzy nimi, ale też ich przeszłymi doświadczeniami i konkretnymi kontekstami, w jakich odbiorca ją uchwyci.

Nie ma gwarancji, że otrzymana przez kogoś informacja jest dokładnie tym, co druga osoba chciała przekazać. Można to zilustrować konkretnym przykładem.

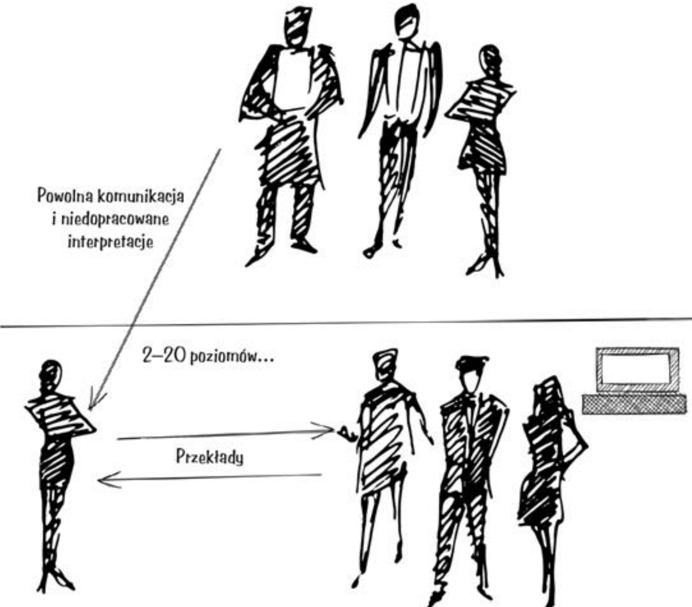
Głuchy telefon

Zabawa w głuchy telefon ukazuje problem związany z pewnymi strukturami komunikacyjnymi. Być może znasz ją pod inną nazwą, ale zasady pozostają takie same. Ludzie ustawiają się w kolejce, a następnie ktoś na jednym jej końcu szepcze na ucho komunikat osobie obok siebie; komunikat jest wyszeptywany kolejno dalej, aż dotrze do ostatniej osoby w kolejce. Na koniec ostatni odbiorca komunikatu wypowiada na głos treść, jaka została mu przekazana, a pierwsza osoba z kolejki ujawnia treść pierwotnego komunikatu. Cała zabawa oczywiście polega na tym, że komunikat ulega skrajnemu zniekształceniu przed dotarciem do końca.

Najciekawsze w tej zabawie i działaniu komunikacji jest to, że zniekształcenie następuje na każdym etapie komunikacji. Każda osoba w kolejce, nawet znajdująca się najbliżej autora pierwotnego komunikatu, słyszy coś, czego nie jest w stanie dokładnie powtórzyć. Im więcej punktów pośrednich, tym większemu zniekształceniu komunikat ulega.

W każdym punkcie przekazu informacji powstaje nowy przekład. Widać wobec tego, że trudności może sprawiać komunikacja odbywająca się pomiędzy zaledwie dwojgiem ludzi. Osiągnięcie jasności i porozumienia nie jest niemożliwe, ale może być trudne.

Kiedy to zjawisko zachodzi w biznesie, nie jest to ani zabawa, ani nie jest to zabawne. Rzecz jasna im bardziej przekaz jest złożony, tym większe jest prawdopodobieństwo wystąpienia większej nieścisłości. Jak widać na rysunku 1.5, często istnieje wiele punktów przekazu informacji. W bardzo dużych organizacjach może to być nawet więcej niż 20 poziomów. Autorzy często słyszą o tak rozbudowanych hierarchiach, że niesłychanie trudne wydaje się zrobienie czegokolwiek w ramach organizacji z jakąkolwiek dokładnością, a programiści — znajdujący się na końcu kolejki — są zdumieni, że cokolwiek działa.



Rysunek 1.5. *Typowa struktura komunikacyjna od kierownictwa najwyższego szczebla po kierowników projektu i programistów*

Trudno dojść do porozumienia

Negatywne uczucia członków zespołu, takie jak obojętność, samozadowolenie, rozczarowanie i demoralizacja, można przezwyciężyć. W tym celu należy pomóc zespołowi w określaniu osiągalnych celów, dostarczać mu nowych, „lekkich" technik, czyli na przykład kształtować go z myślą o usprawnieniu komunikacji, a także angażować się w stopniową, opartą na wartościach restrukturyzację oprogramowania.

Rozbieżności między punktami komunikacji czy nawet stylami komunikacji na każdym poziomie hierarchii mogą jednak pogłębiać przepaść między interesariuszami biznesowymi i technicznymi. Kiedy luka komunikacyjna w obliczu poważnych problemów jest wielka, trudno osiągnąć porozumienie.

Szkodliwy problem pojawia się, gdy kierownictwo techniczne odczuwa, że jest wraz ze swoimi zespołami zagrożone krytyką swojej pracy i widzi, że nadciągają duże zmiany. Zniekształcony przekaz, który słyszy, wskazuje, że istniejący od dawna stan rzeczy jest nie do utrzymania. Jak już niejednokrotnie zauważono na przestrzeni dziejów, człowiek ma ego i często jest silnie przywiązany do tego, co udało mu się osiągnąć dzięki swojej ciężkiej pracy. To silne przywiązanie często określa się mianem „małżeństwa". Kiedy tak silna więź wydaje się zagrożona zerwaniem, zaangażowane strony często przyjmują postawę obronną i nie tylko kurczowo trzymają się tego, *co* zostało zrobione, ale też sposobu, w *jaki* to zostało zrobione. Wyjście poza tę zatwardziałą postawę nie jest łatwe.

Są też osoby z zewnątrz, które zdecydowanie zalecają zmiany niedające się pogodzić z dotychczasowym sposobem prowadzenia działalności. Ten pozorny przeciwnik nie przeszedł przez dziesięciolecia ciężkiej pracy pod presją czasu, którą obwinia się za kłujący w oczy głęboki dług techniczny i entropię. Wszystkie te niekomfortowe spostrzeżenia kumulują się w świadomych myślach kierownictwa technicznego, podgrzewając emocje i zachęcając do posądzania szefostwa firmy niemal o zdradę. Techniczni są przekonani, że zostali wyznaczeni do odstrzału w nagrodę za ciągłe dostarczanie rezultatów w wybitnie niesprzyjających warunkach.

Kiedy liderów technicznych męczy taki niepokój, zazwyczaj dodatkowo go pogłębiają, zwierzając się co najmniej kilku członkom swoich zespołów, którzy zgadzają się z ich obawami. Rzecz jasna ci członkowie zespołów sami zwierzają się innym, więc strach rozprzestrzenia się i wywołuje powszechny opór.

Lecz nie jest to niemożliwe

Cały ten problem najczęściej utrwalany jest przez kulturę firmową znaną jako „my kontra oni". Dzieje się tak z powodu wadliwej struktury komunikacyjnej. Czy dostrzegasz największy problem na rysunku 1.5? Jest nim hierarchia, która rodzi tę mentalność. Rozporządzenia przychodzą z góry, a podwładni wykonują polecenia. Przy zachowaniu takiej hierarchii kierownictwo nie może oczekiwać rezultatów prowadzących do zmian korzystnych dla współpracy.

Taka zmiana musi wynikać z przywództwa, które zaczyna się na najwyższym szczeblu organizacji. Kiedy kierownictwo tego szczebla dostrzega, że hierarchiczne zarządzanie i kontrola są nie do utrzymania, odpowiedzią nie jest zastąpienie jednych kontrolowanych innymi kontrolowanymi.

Kup ksi k Pole ksi k

Przy każdym przedsięwzięciu zespoły odnoszą zdecydowanie większe sukcesy niż jednostki. Dojrzałe drużyny sportowe odnoszą sukcesy poprzez opracowywanie innowacyjnych planów rozgrywek i precyzyjne przekazywanie wszystkim ich zawodnikom poszczególnych zagrywek.

Nie ma zespołowego działania bez zespołu. Komunikacja w zespołach nie odbywa się jednokierunkowo. Każdy członek zespołu może być wystarczająco doświadczony, aby zaproponować coś, co nie znalazło się w wytycznych, albo wskazać, że dane działanie można by przeprowadzić sprawniej dzięki dodaniu określonej czynności lub wyeliminowaniu nieefektywności. Kiedy każdy członek zespołu jest szanowany za swoją kompetencję i doświadczony punkt widzenia, komunikacja staje się znacznie sprawniejsza (rysunek 1.6).

Rysunek 1.6. *Optymalne struktury komunikacyjne biorą się z działania zespołowego*

Oto klucze do optymalnej komunikacji:

- „My” zamiast „my i oni”.
- Przywództwo służebne nie jest poniżej niczyjej godności.
- Trzeba pojąć moc tkwiącą w tworzeniu strategicznych struktur organizacyjnych.
- Nikt nie powinien bać się komunikowania konstruktywnych poglądów.
- Pozytywny wpływ jest kluczowy dla zachęcania ludzi do konstruktywnych działań.
- Formowanie partnerstw biznesowo-technicznych opartych na wzajemnym szacunku jest nieodzowne.
- Głęboka komunikacja, krytyczne myślenie i współpraca są niezbędne dla tworzenia przełomowych, transformacyjnych systemów oprogramowania.

Te strategiczne wzorce behawioralne nie są nowe ani nowatorskie. Są znane od stuleci i są praktykowane przez wszystkie odnoszące sukcesy organizacje.

Prawo Conwaya nie pozostawia w sferze domysłów tego, jak sprawić, aby struktury komunikacyjne organizacji pomagały w uzyskaniu większego dobra. Jak pisze Conway w podsumowaniu swojej pracy:

> Znaleźliśmy kryterium strukturyzowania organizacji projektowych: przedsięwzięcie projektowe powinno być *zorganizowane zgodnie z potrzebami komunikacyjnymi*.

Kup ksi k Pole ksi k

> Ponieważ projekt, który powstaje jako pierwszy, niemal nigdy nie jest najlepszym możliwym rozwiązaniem, konieczna może być zmiana obowiązującej koncepcji systemu. Tym samym *elastyczność organizacji jest ważna dla skutecznego projektowania*.
>
> Należy znajdować sposoby nagradzania kierowników projektów za dbanie o szczupłość i elastyczność organizacji [Conway].
>
> Te koncepcje zostały odzwierciedlone na rysunku 1.6 i przewijają się przez całą tę książkę.

(Nowe) podejście do strategii oprogramowania

Zalecane jest skupienie się na przemyśleniu i zrewidowaniu strategii przed zajęciem się bardziej technicznymi aspektami. Dopóki nie wiemy, jakie strategiczne cele biznesowe należy osiągnąć, nie powinniśmy próbować określać właściwości technicznych systemu. Rozpoczęcie planowania na poziomie systemu będzie miało sens i cel dopiero po pierwszym i ponownym przemyśleniu tej kwestii.

Myślenie

Słynący z licznych godnych wzmianki cytatów George Bernard Shaw wypowiedział się o myśleniu następująco:

> Zakładam, że rzadko myślicie. Mało kto myśli częściej niż dwa lub trzy razy do roku. Dzięki myśleniu raz lub dwa razy w tygodniu udało mi się zdobyć międzynarodową sławę.

Oczywiście wszyscy myślimy codziennie. Życie bez tego nie byłoby możliwe. Zabawna deklaracja Shawa jednak ujawnia ciekawy fakt o ludziach w ogóle. Większa część życia przebiega rutynowo i wiąże się z regularnym stosowaniem swego rodzaju autopilota. Im mniej ludzie muszą myśleć o konkretach, tym mniej są skłonni do świadomego myślenia o tym, co robią. Dlatego właśnie starsze osoby mają tendencję do utraty zdolności poznawczych, jeżeli nie pozostają na stare lata zaangażowane intelektualnie. Shaw pokazuje, że *głębokie myślenie* może nie zdarzać się znów tak często nawet najznamienitszym myślicielom. Brak głębokiego myślenia jest zatem problemem nawet wśród pracowników wiedzy.

Problem z włączaniem się autopilota u takich pracowników polega na tym, że tolerancja oprogramowania na błędy jest niska, w tym zwłaszcza na błędy, na które się przez długi czas nie reaguje. Kiedy indywidualni twórcy oprogramowania nie zachowują ostrożności, zaczynają coraz bardziej opieszale spłacać dług techniczny oprogramowania i przechodzą w tryb, w którym godzą się na nieregulowany wzrost oraz wykonywanie powtarzalnych, doraźnych napraw.

Istnieje również obawa, że programiści zaczną w coraz większym stopniu polegać na tym, co firmy produktowe chcą im sprzedawać, zamiast samodzielnie myśleć w kontekście specyfiki biznesowej przedsiębiorstwa. Szum wokół nowych technologii jest nakręcany przez zewnętrznych aktorów z dużo większą częstotliwością, niż byłoby to możliwe w przypadku wewnętrznych kanałów biznesowych. Takie ciągłe zachwalanie produktów pomija jednak kontekst tego, co ma największe znaczenie w skali lokalnej. Programiści, którzy osiedli na laurach,

Kup ksi k Pole ksi k

mogą chcieć, aby technologia sama rozwiązała problemy za nich. Innym z kolei po prostu brakuje nowych zabawek, które by ich pobudziły. Analogicznie: dynamika leżąca u podstaw strachu przed tym, że coś nas ominie (ang. *fear of missing out*, FOMO), nie jest oparta na głębokim, krytycznym myśleniu.

Jak widać na rysunku 1.7, intensywne myślenie o wszystkim, co wiąże się ze specyfikacją systemu, jest niezbędne do podejmowania właściwych decyzji biznesowych, a następnie niezbędnych pomocniczych decyzji technicznych. Oto kilka sposobów na skontrolowanie motywacji:

- *Co robimy?* Być może niskiej jakości oprogramowanie wypuszczane jest po to, aby wyrobić się w terminie. Zespół nie znajduje się przez to w zbyt dobrym położeniu, aby móc przeprowadzić później refaktoryzację, a dość prawdopodobne jest też, że refaktoryzacji w ogóle nie ma w planach. Możliwe, że zespół naciska na dużą reimplementację z wykorzystaniem nowszej, bardziej popularnej architektury. Nie przeocz faktu, że zaangażowanym w to osobom już się nie udało wprowadzenie pomocnej architektury do istniejących systemów albo że dopuściły do zaniedbań w utrzymywaniu funkcjonującej już architektury.
- *Dlaczego to robimy?* Dochodzące z zewnątrz komunikaty oparte na sprzedaży produktów jako rozwiązań mogą wydawać się bardziej atrakcyjne od wykonywania rozsądnych kroków w zakresie ulepszenia istniejącego oprogramowania, które straciło rację bytu i jedynie podtrzymuje się jego działanie. FOMO i kierowanie się w rozwoju oprogramowania perspektywą wstawienia sobie czegoś atrakcyjnego do CV często sprawniej motywują niż stosowanie dobrych praktyk. Należy upewnić się, czy zastosowanie danej architektury lub technologii jest uzasadnione realnymi potrzebami biznesowymi i technicznymi.
- *Zastanów się nad wszystkim.* Każda pojedyncza nauka musi być poddana krytycznej refleksji z uwzględnieniem za i przeciw. Stanowcze poglądy i donośny głos niczego nie dowodzą. Myślenie w sposób świadomy, jasny, rozległy, głęboki i krytyczny jest niezwykle ważne — to właśnie sprzyja głębokiemu uczeniu się.

Rysunek 1.7. *Bądź liderem w myśleniu. Myśl dużo i rozpoczynaj dyskusje*

Pogoń za głębokim myśleniem rozpoczyna naszą właściwą misję, która polega na przemyśleniu naszego podejścia do rozwoju oprogramowania, aby doprowadzić do uzyskania zróżnicowania strategicznego.

Przemyślenie na nowo

W starożytnej przysiędze Hipokratesa[6] miały padać słowa „po pierwsze nie szkodzić". Wydaje się to celną radą nie tylko w medycynie, ale i w innych dziedzinach, w tym w inżynierii oprogramowania. Starsze, zastane systemy mają pewną wartość. Gdyby jej nie miały, to po prostu by ich nie było. Ciągłe i rozległe użytkowanie systemu sprawia, że jest on aktualnie nie do zastąpienia. O ile specjaliści od oprogramowania często myślą, że pracownicy biznesowi niczego nie rozumieją, o tyle ci ostatni w istocie rzeczy pojmują, że nie wolno niszczyć dziesięcioleci inwestycji w system, który napędzał i wciąż napędza zysk.

Rzecz jasna głębokie zadłużenie i entropia mogą nie być winą tych, którzy aktualnie odpowiadają za podtrzymanie działania systemu, lub tych, którzy zdecydowanie polecają ostateczne zastąpienie go. Mówiąc szczerze, wiele starszych systemów warto odesłać na zasłużoną emeryturę. Jest tak szczególnie w przypadkach, kiedy są one zaimplementowane z użyciem co bardziej archaicznych języków programowania, technologii i urządzeń, których twórcy mają już prawnuki lub ich już z nami nie ma. Jeśli brzmi to jak COBOL wykorzystujący starą bazę danych i działający na mainframe'ie, autorzy bynajmniej nie zaprzeczają podobieństwu.

Istnieją też jednak inne systemy odpowiadające temu opisowi, w tym liczne systemy biznesowe oparte na C/C++. Kiedy powstawały, C/C++ rzeczywiście było lepszym wyborem niż COBOL. Istotną zaletą była mała ilość pamięci wymagana przez programy pisane w C oraz fakt, że dużo oprogramowania tworzono na komputery PC z uwzględnieniem ograniczenia pamięci RAM do 256 – 640 kB. Istnieją też systemy zbudowane na zupełnie przestarzałych i niewspieranych językach oraz technologiach, takich jak FoxPro, zmarginalizowany Delphi oraz praktycznie martwy Visual Basic.

Główny problem z zastępowaniem starego systemu wiąże się z utratą funkcji w procesie wymiany lub zwyczajnym zepsuciem czegoś, co wcześniej działało. Zastąpienie następuje także w toku ciągłej zmiany starego systemu — może i powolnej, ale postępującej. Zmiana nie musi oznaczać wprowadzania nowych funkcji, a może się sprowadzać do codziennego wprowadzania poprawek do kodu i zmian w przechowywanych danych. Zastąpienie ruchomego celu innym ruchomym celem jest trudne. Nie wspominamy tu nawet o tym, że oprogramowanie znajduje się w takim, a nie innym stanie, ponieważ nie poświęcono mu uwagi, na jaką zasługiwało i jakiej potrzebowało. Trudno zatem znaleźć w sobie olbrzymie pokłady troski, kiedy cel jest ruchomy, a ludzie ciągle starają się w niego trafić.

Fakt, że system ma być słusznie zastąpiony z wykorzystaniem nowoczesnych architektur, języków programowania i technologii, nie ułatwia tego zadania. Wielu dochodzi do wniosku, że jedynym wyjściem jest rozebranie aktualnej implementacji i napisanie nowej. Często zdarza się, że osoby podejmujące takie wyzwania proszą o kilka miesięcy wolnych od zakłóceń

[6] Aktualność i zasadność przysięgi Hipokratesa we współczesnych czasach, podobnie jak to, gdzie rzeczywiście po raz pierwszy padły słowa „po pierwsze nie szkodzić", nie jest tutaj przedmiotem dyskusji. Wielu lekarzy wciąż uznaje ważność przysięgi i wskazanej tu maksymy.

Kup ksi k Pole ksi k

wynikających ze zmian, aby przeprowadzić taką akcję. Oznaczałoby to jednak zatrzymanie ruchomego celu w jednym miejscu na kilka miesięcy, a — jak już zauważono — system najprawdopodobniej będzie w tym czasie wymagał wprowadzenia poprawek w kodzie i danych. Czy wobec tego należy się wstrzymywać z tym przez nieokreślony czas?

Kiedy nie ma innego wyboru

Jeden z autorów brał udział w przedsięwzięciu, w ramach którego platformy wymykały się wdrożeniu. Wyobraź sobie na przykład przeniesienie rozległego systemu z graficznym interfejsem użytkownika wdrożonego w MS-DOS do Microsoft Windows API. Jedną z takich naprawdę problematycznych kwestii, o których nikt nie myśli, dopóki nie ugrzęźnie w miejscu, jest to, że dwa API mogą transponować parametry. W różnych API współrzędne X, Y interfejsu mogą ulegać zmianie. Przeoczenie choćby i jednej z takich zmian może stać się źródłem ogromnych problemów, które trudno wytropić. Nadrzędnym powodem złożoności problemu było niedostateczne bezpieczeństwo zasobów pamięci w programach C/C++, gdzie nieprawidłowe odwołania prowadziły do nieprawidłowego nadpisywania pamięci, czasami w zupełnie inny sposób za każdym razem. Wskutek tego trzeba było się liczyć z dziwacznymi naruszeniami pamięci, następującymi na wiele tajemniczych sposobów.

Rzecz jasna nie jest to typowy problem spotykany przy dzisiejszych modernizacjach — nowoczesne języki programowania na ogół zapobiegają występowaniu tego typu błędów. Niemniej jednak nieprzewidywalne problemy zawsze mogą się pojawić. Radzenie sobie z takimi nieplanowanymi utrudnieniami może pożreć lwią część tych „kilku miesięcy” przeznaczonych w założeniu na „rozebranie aktualnej implementacji i napisanie nowej”. To zawsze jest trudniejsze i trwa dłużej, niż zakładasz.

Gdzie można się tutaj doszukiwać przemyślenia wszystkiego na nowo? Wygląda na to, że zupełne pozbycie się istniejącego oprogramowania może doprowadzić do poważnych szkód. Takie podejście do rozwiązania problemu jest odruchową reakcją na rozległe problemy, a często wywołuje jeszcze większe problemy czy też dwa zbiory zupełnie odrębnych, ale równie olbrzymich problemów. Wielka kula błota jako norma w przedsiębiorczości prowadzi do paraliżu konkurencyjnego — przede wszystkim nie należy jednak szkodzić, a pacjent musi być w stanie oddychać, żeby możliwe było przeprowadzenie przywracającej go do zdrowia operacji. Musimy znaleźć sposób na przeprowadzenie reimplementacji, ale nie powinien to być skok na głowę z wielkim rozpryskiem. Temu rozpryskowi można zapobiec za pomocą specjalnych środków i technik.

Nie uwzględnia się tutaj tworzenia nowych okazji do uczenia się. Jeśli poprzestaniemy na przepisaniu w C# dużego systemu, który pierwotnie został zaimplementowany w Visual Basicu, ze strategicznego punktu widzenia niczego się nie nauczymy. Jeden z klientów zauważył na przykład podczas wymiany starszego systemu COBOL, że 70% opracowanych w ciągu 40 lat reguł biznesowych stało się przestarzałych. Były one wciąż obecne w kodzie COBOL, a ich obsługa zwiększała obciążenie poznawcze. Wyobraźmy sobie, że nie pozyskaliśmy tej informacji

Kup ksi k Pole ksi k

i zamiast tego poświęciliśmy czas i wysiłek na przeniesienie wszystkich tych reguł biznesowych z języka COBOL na nowoczesną architekturę, język programowania i zestaw technologii. Ta transformacja była sama w sobie kompleksowym, rozłożonym na wiele lat przedsięwzięciem, nawet bez uwzględnienia ogromnego wolumenu zupełnie niepotrzebnych przekształceń.

Poniższe pytania, jako rozszerzenie wskazanych wcześniej sposobów sprawdzania motywacji, wskazują na potrzebę zdobywania istotnej strategicznej wiedzy:

- *Jakie są cele i strategie biznesowe?* Każda funkcja oprogramowania tworzonego w ramach inicjatywy strategicznej powinna być bezpośrednio związana z podstawowym celem biznesowym. Aby to było możliwe, określ (1) cel biznesowy, (2) docelowy segment rynkowy (jednostki lub grupy), na który należy wywrzeć wpływ, aby osiągnąć ten cel, oraz (3) wpływ, jaki trzeba wywrzeć na docelowy segment rynkowy. Dopóki nie zrozumie się koniecznych wpływów, nie ma się możliwości wskazania wymaganej funkcjonalności oprogramowania ani zakresu konkretnych wymagań. Narzędzia służące do identyfikowania celów i wpływów strategicznych opisane są w dalszej części książki.

- *Dlaczego tego nie robimy?* Przy podejmowaniu decyzji strategicznych należy uwzględnić jeszcze jedno ważne pojęcie z zakresu technologii: nie będziesz tego potrzebować (ang. *You Aren't Gonna Need It*, YAGNI). Maksyma ta miała pomagać zespołom unikać rozwijania niepotrzebnych aktualnie funkcji biznesowych, ku czemu są dobre powody. Poświęcanie czasu i pieniędzy oraz podejmowanie ryzyka w celu dostarczania niepotrzebnego oprogramowania to zły pomysł. Niestety, powoływanie się na YAGNI stało się powszechnym sposobem na spychanie na bok każdego przeciwnego punktu widzenia. Używanie YAGNI w charakterze weta nie sprzyja pozyskiwaniu lojalności zespołu ani nie tworzy przełomowych możliwości uczenia się. Czasami niewdrożenie pewnych funkcji, które „nie są potrzebne", jest błędem na ogromną skalę. Jeśli przełom, który może doprowadzić do zróżnicowania innowacyjnego, jest natychmiast odrzucany, to prawdopodobnie jest to problem ze zdolnością do głębokiego myślenia i rozpoznawania możliwości lub zagrożenia ich utratą. Całkowita niechęć do zapewnienia przestrzeni dla przyszłych dyskusji na ten temat jest cechą pozwalającą na wskazanie najsłabszych myślicieli w zespole.

- *Czy możemy spróbować czegoś nowego?* Zespoły mogą być zgodne lub niezgodne co do tego, co może się sprawdzić na ich rynku docelowym. Przeważnie niemożliwe jest całkowite przewidzenie reakcji rynku na daną strategię. Dokładna ocena reakcji rynku wymaga umożliwienia mu zweryfikowania pomysłów biznesowych. Eksperymentowanie może być jedynym sposobem na zrozumienie rzeczywistych możliwości i ograniczeń strategii. Nie zawsze jednak łatwo jest wykroczyć myślami poza ugruntowany model mentalny na potrzeby wypróbowywania nowości. „Ludziom niesłychanie trudno przychodzi zdanie sobie sprawy z tego, że są przykuci do jakiegoś modelu, zwłaszcza jeśli jest on podświadomy bądź tak silnie wpleciony w kulturę lub ich oczekiwania, że nie potrafią już dostrzec, jak bardzo ich tłamsi" [Brabandère].

- *Jakie są wymagania dotyczące poziomu usług?* Po zaznajomieniu się z rozsądnym zestawem strategicznych celów biznesowych zespoły zaangażowane w prace mogą przystąpić do identyfikacji koniecznych decyzji architektonicznych, które należy podjąć. Wybór decyzji

Kup ksi k Pole ksi k

zależny jest od wymagań dotyczących poziomu usług. Zespoły nie powinny zbyt szybko decydować się na rozwiązania, ponieważ opóźnianie podejmowania decyzji dotyczących szczegółów architektury często jest korzystne. Na przykład nawet jeśli zespoły są przekonane, że konieczne jest zastosowanie architektury mikrousług, opóźnienie wdrożenia usług rozdzielonych siecią komputerową może pomóc zespołowi skupić się na rzeczywistych czynnikach biznesowych, zamiast zbyt wcześnie starać się poradzić sobie z rozproszonym obciążeniem obliczeniowym. (Zob. podrozdział „Wdrożenie na koniec" w rozdziale 2. „Podstawowe narzędzia strategicznego uczenia się").

Przemyślenie działań na nowo jest kluczowym krokiem i wydaje się z wszech miar słuszne. Myślenie wielowymiarowo i krytycznie oraz ponowne przemyślenie sytuacji, aby odejść od sztampowego punktu widzenia na rzecz nowej, strategicznej perspektywy, niesie ze sobą korzyści.

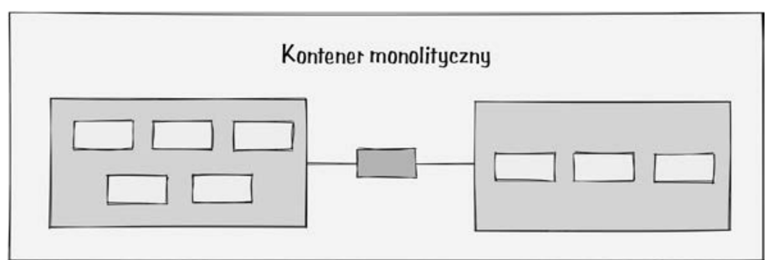
Nie należy jednak wyciągać wniosku, że wszystkie istniejące monolity można z samego założenia klasyfikować jako wielkie kule błota. O ile rzeczywiście można tak powiedzieć o przeważającej większości z nich, takiej oceny należy podejmować się bardzo ostrożnie. Temat ten poruszamy poniżej.

Czy monolity są złe?

W ciągu kilku ostatnich lat pojęcia *monolit* i *monolityczny* używane w odniesieniu do oprogramowania nabrały bardzo negatywnych konotacji. Mimo to fakt, że przeważająca większość istniejących monolitycznych systemów przyjęła formę wielkiej kuli błota, nie oznacza, że zawsze musi to nastąpić. To nie monolit jest problemem — jest nim błoto.

Pojęcie **monolitu** może zwyczajnie oznaczać, że oprogramowanie całej aplikacji lub systemu jest osadzone w kontenerze zaprojektowanym z myślą o przechowywaniu więcej niż jednego podsystemu. Kontener monolityczny często obejmuje wszystkie lub większość podsystemów całej aplikacji lub systemu. Ponieważ każda z części systemu zawarta jest w jednym kontenerze, określa się go mianem **samodzielnego**.

Wewnętrzna architektura monolitu może być zaprojektowana tak, aby komponenty różnych podsystemów były od siebie odizolowane, ale możliwe jest też zapewnienie komunikacji i wymiany informacji pomiędzy podsystemami. Rysunek 1.8 przedstawia oba podsystemy z rysunku 1.1, lecz zawarte w monolitycznym kontenerze.



Rysunek 1.8. *Kontener monolityczny ukazujący część całego systemu. Widoczne są tutaj jedynie dwa z potencjalnie kilku podsystemów składających się na całość*

Przy rysunku 1.1 założyliśmy, że oba podsystemy są od siebie oddzielone w ramach dwóch procesów i że komunikują się przez sieć. Tamten diagram wskazywał na system rozproszony. Na rysunku 1.8 te same dwa podsystemy są ze sobą fizycznie połączone w ramach jednego procesu i prowadzą wymianę informacji za pomocą prostych mechanizmów wewnątrzprocesowych, takich jak metody i funkcje języka programowania.

Nawet jeśli architektura systemu ma ostatecznie przyjąć postać mikrousług, rozpoczęcie tworzenia systemu w formie monolitu niesie pewne korzyści. Brak sieci łączącej podsystemy może zapobiec wielu problemom, z którymi radzenie sobie na wczesnym etapie jest zbędne i kontrproduktywne. Co więcej, skorzystanie z monolitu jest dobrym sposobem na wykazanie się zaangażowaniem w luźne sprzężanie podsystemów, kiedy łatwiej byłoby pozwolić na sprzężenie ścisłe. Jeśli plan zakłada przekształcenie architektury w mikrousługową, ostatecznie można się dzięki temu przekonać, na ile luźne rzeczywiście jest to sprzężenie.

Choć niektórzy są przeciwni wykorzystywaniu architektury monolitycznej na wczesnym etapie prac, które docelowo mają doprowadzić do uzyskania architektury mikrousług, zachęcamy do wstrzymania się od oceny przed zapoznaniem się z omówieniem tego zagadnienia w częściach II i III tej książki.

Czy mikrousługi są dobre?

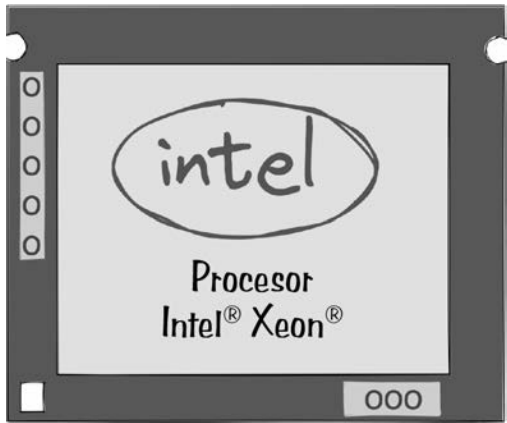
Pojęcie **mikrousługi** ma wiele znaczeń. Według jednej definicji mikrousługa powinna składać się maksymalnie ze 100 linijek kodu. Według innej linijek może być maksymalnie 400. Jeszcze inna twierdzi, że linijek kodu może być góra 1000. Z wszystkimi tymi próbami zdefiniowania pojęcia mikrousług, odnoszącymi się raczej do samej nazwy, wiąże się co najmniej kilka problemów. „Mikro” często wydaje się wskazywać na wielkość, ale co ten człon właściwie oznacza?

Kiedy używa się przedrostka „mikro” do opisu procesora komputerowego, używa się pełnej nazwy: *mikroprocesor*. Podstawowa zasada działania mikroprocesora polega na tym, że wszystkie funkcje procesora wykonywane są przez jeden lub kilka układów scalonych. Przed narodzinami i rozpowszechnieniem się mikroprocesorów komputery zazwyczaj bazowały na szeregu płytek drukowanych z wieloma układami scalonymi.

Należy jednak zauważyć, że termin *mikroprocesor* nie wiąże się z pojęciem wielkości, tak jakby arbitralnie określona liczba układów scalonych lub tranzystorów była właściwa lub nie. Rysunek 1.9 ukazuje przykład — jeden z najpotężniejszych dostępnych procesorów jest mikroprocesorem. W 28-rdzeniowym Xeon Platinum 8180 znajduje się 8 miliardów tranzystorów. Wydany w roku 1971 Intel 4004 składał się z 2250 tranzystorów. Oba te procesory są mikroprocesorami.

Limity mikroprocesorów są z reguły określane na podstawie ich przeznaczenia. Oznacza to, że niektóre ich rodzaje zwyczajnie nie muszą zapewniać takiej mocy, jaką oferują inne. Mogą być używane w małych urządzeniach o ograniczonych zasobach energii, więc ich pobór musi być odpowiednio mniejszy. Ponadto, kiedy moc pojedynczego mikroprocesora — nawet niesamowicie rozbudowanego — nie jest wystarczająca w danych warunkach obliczeniowych, komputery wyposażane są w większą ich liczbę.

Kup ksi k Pole ksi k



Rysunek 1.9. *Intel Xeon jest jednym z najpotężniejszych nowoczesnych mikroprocesorów. Nikt nie twierdzi, że ma „za dużo układów i tranzystorów" i nie jest przez to mikroprocesorem*

Kolejnym problemem z nakładaniem arbitralnego limitu na liczbę linijek kodu mikrousługi jest to, że od samego języka programowania zależna jest liczba linijek kodu potrzebnych do obsłużenia określonej funkcji systemu. Limit 100-linijkowy oznacza coś innego w przypadku Javy i Ruby, ponieważ Java na ogół wymaga 33% więcej kodu niż Ruby. Z kolei z zestawienia Javy z Clojure wynika, że ten pierwszy język wymaga 360% więcej linijek kodu.

Ponadto, co jeszcze bardziej istotne, tworzenie maleńkich mikrousług niesie ze sobą pewne wady:

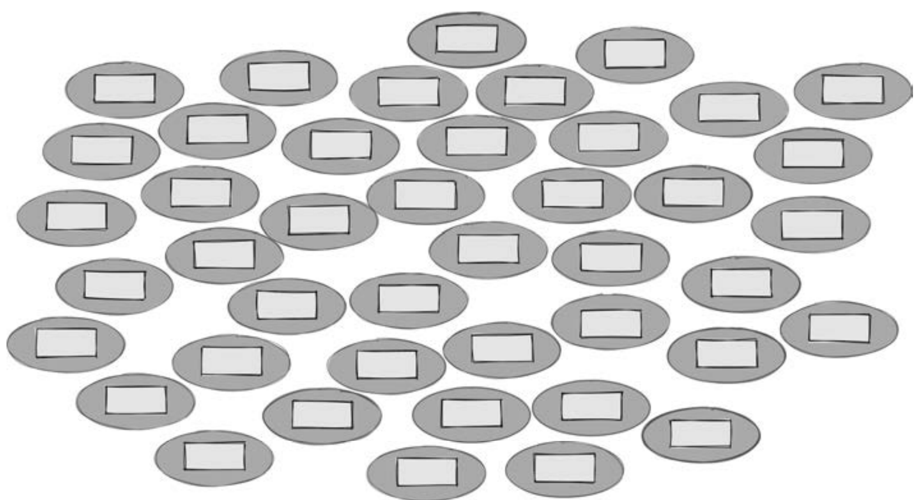
- Liczba mikrousług może sięgnąć setek, tysięcy, a nawet dziesiątek tysięcy.
- Brak zrozumienia zależności prowadzi do nieprzewidywalności zmian.
- Nieprzewidywalność zmian skutkuje brakiem zmian lub wycofywania komponentów.
- Powstaje więcej mikrousług o podobnej funkcjonalności (metodą „kopiuj-wklej").
- Koszt utrzymania rozrastających się, lecz często przestarzałych mikrousług wzrasta.

Te główne wady wskazują, że w rezultacie mało prawdopodobne jest uzyskanie oczekiwanego, bardziej przejrzystego, rozproszonego systemu i autonomii. Dzięki przedstawionemu w tym rozdziale kontekstowi, problem powinien być jasny. Tworzenie wielu maleńkich mikrousług prowadzi do wystąpienia takich samych warunków jak w przypadku monolitycznej wielkiej kuli błota, jak widać na rysunku 1.10. Nikt nie rozumie systemu, który cierpi na te same przypadłości, co tego typu monolit: bezładną strukturę, niekontrolowany rozrost, powtarzalne i doraźne naprawy, bezładne przekazywanie informacji, globalny lub powielony status wszystkich ważnych informacji.

Można zastosować pewne rozwiązania i podjąć dodatkowe wysiłki, aby uniknąć tego rezultatu, ale z reguły nie polega to na wdrażaniu oddzielnego kontenera dla każdej mikrousługi.

O mikrousługach najlepiej myśleć nie w kontekście ich wielkości, tylko przeznaczenia. Mikrousługa jest niewielka w porównaniu z monolitem, ale zalecamy jednak unikanie tworzenia mikroskopijnych mikrousług. Te zagadnienia omawiamy dalej, w części II tej książki.

Kup ksi k Pole ksi k



Rysunek 1.10. *Duża liczba maleńkich mikrousług przyjmuje formę rozproszonej wielkiej kuli błota*

Nie obwiniaj Agile

W filmie *Jeśli dziś wtorek, to jesteśmy w Belgii* z 1969 roku przewodnik oprowadza grupy Amerykanów po Europie w szaleńczym tempie. Z perspektywy rzeczywistego zwiedzania jest to klasyczny przykład rutynowego odhaczania kratek.

To samo może stać się ze zwinnym wytwarzaniem oprogramowania. „Mamy 10 rano i stoimy sobie w grupie. To chyba znaczy, że jesteśmy zwinni". W przypadku codziennych stand-upów taka rutyna zamienia wartościową komunikację projektową w pozbawioną sensu ceremonię. Lektura wątku *Agile: Methodology or Framework or Philosophy* („Agile: metodyka, framework czy filozofia") na *Scrum.org* naprawdę pomaga człowiekowi otworzyć oczy. W chwili pisania tego tekstu pierwotny post miał 20 odpowiedzi, w większości wyrażających różne opinie[7]. Rozsądne wydaje się zadanie sobie pytania: dlaczego powinno to mieć znaczenie?

Zwinne tworzenie oprogramowania przyciąga ostatnimi czasy dużo krytyki. Większość z niej ukierunkowana jest na konkretną metodykę zarządzania projektem, z której certyfikację można uzyskać już po kilku dniach szkolenia, ponieważ obie te rzeczy są często ze sobą utożsamiane. To przykry stan rzeczy, jeśli weźmie się pod uwagę, co zwinne wytwarzanie oprogramowania powinno oznaczać. Znaczna część branży, która twierdzi, że stosuje metodykę zwinną lub działa zwinnie, nie potrafi tak naprawdę zdefiniować tego pojęcia, co widać na przykładzie Scrum.org.

Filozofia Agile pierwotnie nigdy nie obiecywała, że przekształci słabych programistów w dobrych. Czy w ogóle składała ona jakieś obietnice? Istnieje sposób myślenia o tworzeniu oprogramowania zwinnie (*Manifest programowania zwinnego*) i ma on swoją historię [Cockburn]. Jak już zauważono, ta metodyka nie sprawia, że programiści przyjmują taki sposób myślenia.

[7] Zważywszy, że przy trzech opcjach permutacji jest sześć, pojawienie się konkretnie 20 różnych odpowiedzi wydaje się nawet dziwniejsze od faktu, że tyle odpowiedzi w ogóle istnieje.

Kup ksi k Pole ksi k

Rozważmy pewien problem. Idea zwinnego wytwarzania oprogramowania została sprowadzona do sporów o to, czy należy to podejście nazywać „Agile”, „agile”, czy „podejściem zwinnym”. Właściwie moglibyśmy ściągnąć na siebie ogień krytyki za samo użycie słowa „podejście”. Co za tym idzie, od tej pory nie będziemy już pisać o „tym” per „to”, lecz będziemy używać zapisu #agile, który ma reprezentować każde możliwe zastosowanie. Niezależnie od tego, jak kto woli nazywać #agile, twórcy oprogramowania muszą starać się wyciągnąć z #agile więcej, niż #agile wyciąga z nich.

Rozważmy jeszcze jeden problem. Wokół dość prostych koncepcji narosła ogromna złożoność. Na przykład pojęcia i etapy #agile są przedstawiane w formie map transportu publicznego. Co najmniej jedna z renomowanych firm konsultingowych przedstawia swoje podejście #agile w postaci mapy systemu metra w Nowym Jorku lub Londynie. Choć podróżowanie w dowolne miejsce rozległej sieci komunikacyjnej nie jest skrajnie skomplikowane, mało komu chciałoby się z codzienną lub cotygodniową regularnością wypróbowywać wszystkie możliwe trasy dojazdu do pracy z rana i do domu wieczorem.

To wszystko jest bardzo niefortunne. Wielu ludzi zniekształca #agile i oddala je od korzeni, a inni zwyczajnie przyjmują naiwne podejście. Stosowanie #agile powinno być znacznie prostsze. Praca z tym podejściem powinna sprowadzać się do czterech rzeczy: współpracy, dostarczania, refleksji i ulepszania [Cockburn-Forgiveness].

Przed wkroczeniem na pogmatwaną ścieżkę zespoły powinny nauczyć się, jak dojechać do pracy i wrócić do domu w czterech podstawowych krokach:

1. *Określenie celów.* Cele są szersze od pojedynczych zadań roboczych. Osiąganie ich wymaga współpracy w celu określenia wpływu, jaki oprogramowanie ma wywrzeć na konsumencie. Takie wpływy pozytywnie zmieniają zachowania konsumentów. Konsumenci rozpoznają w nich to, czego potrzebują, zanim jeszcze stają się świadomi własnych potrzeb.

2. *Definiowanie krótkich iteracji i wdrażanie.* Iteracja jest procedurą, w ramach której powtórzenie sekwencji działań przynosi wyniki coraz bardziej zbliżone do pożądanego rezultatu. Zespół wspólnie określa jej kształt. Ze względu na presję towarzyszącą realizacji projektu, nieuniknione przerwy i inne rozpraszające czynniki, w tym końce dni i tygodni, zespół powinien ograniczyć liczbę zadań tak, aby móc je z łatwością zapamiętać i wykonać.

3. *Wdrażanie przyrostów po wypracowaniu właściwej wartości.* Przyrost jest czynnością lub procesem zwiększania, zwłaszcza ilości lub wartości, uzyskiwaniem lub dodawaniem czegoś czy też stopniem lub ilością wprowadzanych zmian. Jeśli dostarczenie wartości nie jest możliwe w ciągu jednego dnia pracy, to należy chociaż osiągnąć przyrost ku docelowej wartości. Zespoły powinny być w stanie dostarczyć wartość w wyniku jednego lub dwóch dni iteracji.

4. *Ocena wyników, zapisanie długu, powrót do punktu 1.* Należy się teraz zastanowić nad tym, co udało się osiągnąć (lub czego nie), z zamiarem poprawy. Czy przyrost wywarł zamierzony i istotny wpływ? Jeśli nie, zespół może zwrócić się ku innemu zbiorowi iteracji z przyrostami o odmiennej wartości. W takim przypadku odnotuj, co zespół uznaje za powody sukcesu. Nawet po dostarczeniu kluczowej wartości normą jest to, że przyrostowy wynik nie daje zespołowi poczucia pełnego sukcesu. W takich przypadkach wdrożenie prowadzi do zgromadzenia większej wiedzy i jaśniejszego zrozumienia przestrzeni

Kup ksi k Pole ksi k

problemu. Jednocześnie iteracja jest ograniczona czasowo i nie pozostawia wystarczająco dużo czasu na natychmiastowe przemodelowanie lub zrefaktoryzowanie bieżących wyników. Należy to zapisać jako dług. Dług można obsłużyć w kolejnych iteracjach, co prowadzi do zwiększenia dostarczanej wartości.

Planowanie ze zbyt wielkim wyprzedzeniem prowadzi do konfliktów w zakresie celów i realizacji. Zbyt szybkie działanie może prowadzić do celowego odwracania wzroku od długu lub zapominania o rejestrowaniu go. Zespół znajdujący się pod dużą presją może przestać interesować się długiem prędzej niż później.

Tych kilka kroków wskazanych w powyższym, krótkim przeglądzie podstaw #agile może pomóc zespołom zajść daleko. Eksperymentowanie pozwala na przyjęcie takiego sposobu myślenia i to właśnie na tym powinno się skupiać w #agile. Z #agile rzeczywiście można wyciągnąć więcej, niż trzeba na #agile poświęcić.

Wyrwać się z błota

Każda firma, która utknęła w wielkiej kuli błota i zmuszona jest improwizować z użyciem złożonych technik i technologii, musi się z niej wyrwać i wejść na właściwą drogą. Nie ma tutaj prostych rozwiązań ani złotych środków. Istnieją jednak sposoby, które mogą okazać się pomocne.

Głęboko zadłużony system oprogramowania, który prawdopodobnie osiągnął maksymalny poziom entropii, potrzebował lat, a nawet dziesięcioleci, aby jego solidna funkcjonalność uległa erozji i odeszła od pierwotnego zamysłu. Wyjście z tego bałaganu będzie wymagało czasu. Mimo to wprowadzanie istotnych zmian nie jest stratą czasu i pieniędzy. U podstaw tego stwierdzenia leżą dwa czynniki odnoszące się do złej decyzji o kontynuowaniu inwestowania środków w przegraną sprawę:

- *Eskalacja zaangażowania.* Jest to wzorzec zachowania ludzkiego, w ramach którego jednostka lub grupa mierząca się z coraz bardziej negatywnymi rezultatami danej decyzji, działania lub inwestycji mimo wszystko nie zmienia swojego zachowania. Podmiot zachowuje się w sposób irracjonalny, ale zgodny z wcześniejszymi decyzjami i działaniami [EoC].
- *Efekt utopionych kosztów.* Koszt utopiony to zapłacona w przeszłości kwota, która nie ma już znaczenia dla decyzji dotyczących przyszłości. „Koszty utopione rzeczywiście wpływają na ludzkie decyzje, ponieważ ludzie wierzą, że inwestycje (np. takie koszty) uzasadniają dalsze wydatki. Ludzie wykazują się »większą skłonnością do kontynuowania przedsięwzięcia po zainwestowaniu pieniędzy, wysiłku lub czasu«. Takie zachowanie można nazwać łataniem dziur pieniędzmi przy jednoczesnym odmawianiu zmniejszania strat” [KosztyUtopione].

Nie oznacza to, że zachowanie jakiejś części zastanego systemu zawsze jest tożsame z uleganiem efektowi utopionych kosztów. Rzecz w tym, że utrzymywanie istniejącego systemu w aktualnym stanie, wraz z jego głębokim zadłużeniem i niemal maksymalną entropią, jest przegranym rozwiązaniem z perspektywy zarówno emocjonalnej, jak i finansowej.

Kup ksi k Pole ksi k

Czas nie stoi w miejscu, a zmiany nie ustają, gdy zespoły toczą heroiczny bój z wielką, brązową bestią. Poruszanie się naprzód w miarę upływu czasu przez błoto w atmosferze ciągłych zmian i bez dalszego grzęźnięcia w miejscu jest absolutną koniecznością. Sukces w takich warunkach zależy bardziej od nastawienia niż od architektury przetwarzania rozproszonego. Pozytywne nastawienie rozwija się przez poczucie pewności, a pozostała część tej książki zawiera szereg narzędzi i technik pozwalających zbudować pewność siebie niezbędną do osiągnięcia strategicznych innowacji.

Podsumowanie

W tym rozdziale omówiono znaczenie *innowacji* jako sposobu na osiągnięcie głównego celu biznesowego, jakim jest uzyskiwanie wyróżniającego się oprogramowania. Dążenie do nieustannego doskonalenia w ramach transformacji cyfrowej jest najlepszym wyborem w czasach, w których „oprogramowanie pożera świat". Wprowadzono pojęcie architektury oprogramowania i omówiono rolę, jaką odgrywa w każdej firmie. W rozdziale przeanalizowano, w jaki sposób prawo Conwaya kształtuje ścieżki komunikacyjne w obrębie organizacji i zespołów oraz jak wpływa na tworzone przez nie oprogramowanie. W omówieniu znaczenia komunikacji kwestia *wiedzy* odegrała pierwszoplanową rolę. Wiedza jest jednym z najważniejszych zasobów każdej firmy. Aby uzyskać najlepsze możliwe wyniki pracy nad oprogramowaniem, wiedzę milczącą należy przekształcić we wspólną. Dzielenie się wiedzą jest niemożliwe bez odpowiednich ścieżek komunikacyjnych, podobnie jak uzyskanie przewagi konkurencyjnej. W ostatecznym rozrachunku słaba komunikacja przekłada się na niekompletną wiedzę i źle modelowane oprogramowanie — wówczas liczyć można co najwyżej na wielką kulę błota. Na koniec skupiliśmy się na pierwotnym sposobie myślenia #agile i tym, w jaki sposób może ono pomóc zespołom przełamać impas i skupić się na właściwych celach.

Oto najważniejsze wnioski płynące z tego rozdziału:

- Innowacja jest najważniejszym aspektem transformacji cyfrowej. Innowacja prowadzi do korzystnego odróżnienia się od konkurencji, więc tym samym powinna być strategicznym celem każdej firmy.
- Architektura oprogramowania musi wspierać wprowadzanie nieuchronnych zmian bez ponoszenia wysokiego kosztu i wysiłku. Alternatywą dla dobrej architektury jest zła architektura, a ta ostatecznie przekształca się w niearchitekturę.
- Wielka kula błota często powstaje jako rezultat niesprawnej komunikacji, która nie może prowadzić do głębokiego uczenia się i tworzenia wspólnej wiedzy.
- Członkowie organizacji muszą wymieniać się wiedzą w toku otwartej i pełnej niuansów komunikacji, umożliwiającej tworzenie przełomowych innowacji.
- Monolity niekoniecznie są złe, a mikrousługi nie są koniecznie dobre. Wybór architektury powinien być wynikiem świadomej decyzji.

Kup ksi k Pole ksi k

Rozdział 2. przedstawia narzędzia strategicznego uczenia się, które mogą pomóc w usprawnianiu komunikacji i podnieść poziom kultury organizacyjnej. Zastosowanie ich pozwoli Ci nauczyć się, jak podejmować świadome decyzje na podstawie eksperymentów, i dowiedzieć się, w jaki sposób mogą one wpłynąć na powstałe oprogramowanie i jego architekturę.

Źródła

[**ANW**] A.N. Whitehead, *Technical Education and Its Relation to Science and Literature*, „Mathematical Gazette", 9, nr 128, 1917, s. 20 – 33.

[**a16z-CloudCostParadox**] *https://a16z.com/2021/05/27/cost-of-cloud-paradoxmarket--cap-cloud-lifecycle-scale-growth-repatriation-optimization/.*

[**BBoM**] *https://en.wikipedia.org/wiki/Big_ball_of_mud.*

[**Brabandère**] Luc de Brabandère i Alan Iny, *Thinking in New Boxes: A New Paradigm for Business Creativity*, Random House, New York 2013.

[**Cockburn**] *https://web.archive.org/web/20170626102447/http://alistair.cockburn.us/How+I+saved+Agile+and+the+Rest+of+the+World.*

[**Cockburn-Forgiveness**] *https://www.youtube.com/watch?v=pq1EXK_yL04* (w języku francuskim, angielskim i hiszpańskim).

[**Conway**] *http://melconway.com/Home/Committees_Paper.html.*

[**Cunningham**] *http://wiki.c2.com/?WardExplainsDebtMetaphor.*

[**Entropia**] *https://pl.wikipedia.org/wiki/Entropia.*

[**EoC**] *https://en.wikipedia.org/wiki/Escalation_of_commitment.*

[**Jacobson**] *https://en.wikipedia.org/wiki/Software_entropy.*

[**KosztyUtopione**] *https://pl.wikipedia.org/wiki/Koszty_utopione.*

[**LAMSADE**] Pierre-Emmanuel Arduin, Michel Grundstein, Elsa Negre i Camille Rosenthal-Sabroux, „Formalizing an Empirical Model: A Way to Enhance the Communication between Users and Designers", *IEEE 7th International Conference on Research Challenges in Information Science*, 2013, s. 1 – 10; doi: *10.1109/RCIS.2013.6577697.*

[**Manifest**] *https://agilemanifesto.org/iso/pl/manifesto.html.*

[**Polanyi**] M. Polanyi, *Sense-Giving and Sense-Reading*, „Philosophy: Journal of the Royal Institute of Philosophy", 42, nr 162, 1967, s. 301 – 323.

Kup ksi k Pole ksi k

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

PRZEPIS NA SUKCES TKWI W DOSKONAŁOŚCI PODEJŚCIA ARCHITEKTONICZNEGO

Przedsiębiorstwo, jeśli ma osiągać satysfakcjonujące wyniki biznesowe, musi używać dobrego oprogramowania. Aby jednak zapewnić firmie pozycję lidera i decydować o przyszłości branży, trzeba czegoś więcej: odwagi we wprowadzaniu i wdrażaniu innowacji. Innowacje te powinny być wspierane przez inteligentne decyzje architektoniczne ukierunkowane na cele firmy, osiągane wyniki i zapewnienie sobie przewagi konkurencyjnej w przyszłości. Niestety, podczas projektowania oprogramowania architekci często kierują się przyzwyczajeniami albo aktualnie obowiązującą modą.

Ta książka jest przeznaczona dla kadry kierowniczej najwyższego szczebla i dla osób sterujących rozwojem oprogramowania w firmie. Ma pomóc w zrozumieniu problemów strategicznych, z jakimi te osoby się mierzą, a także ułatwić wybór najlepszego rozwiązania architektonicznego. W książce opisano, kiedy zdecydować się na rozproszone mikrousługi czy dobrze zmodularyzowane monolity, a kiedy na usługi będące połączeniem obu rozwiązań. Dokładnie wyjaśniono, w jak dużym stopniu wyważone decyzje architektoniczne umożliwiają maksymalizację wartości i innowacyjności, dostarczanie łatwych do rozwijania systemów i unikanie kosztownych błędów. Nie zabrakło tu również praktycznych wskazówek, jak tworzyć dobrze zaprojektowane monolity, które można bez problemu utrzymywać i rozwijać, a także jak stopniowo przekształcać starsze systemy w prawdziwie efektywne mikrousługi.

Najciekawsze zagadnienia:

- łączenie planowania architektury z wprowadzaniem innowacji w firmie
- problemy komunikacyjne a eksperymentowanie z innowacjami
- praktyczne podejście do strategicznych inwestycji
- najlepsze style architektoniczne
- wybór między systemem monolitycznym a mikrousługami
- przekształcanie monolitów w mikrousługi

Vaughn Vernon jest przedsiębiorcą, programistą i architektem, a także ekspertem w dziedzinie DDD oraz architektury i programowania reaktywnego. Prowadzi konsultacje i szkolenia. Jest również autorem książek technicznych.

Tomasz Jaskuła jest dyrektorem technicznym i współzałożycielem Luteceo, paryskiej firmy konsultingowej. Od ponad 20 lat pracuje jako profesjonalny programista i architekt, a jego celem jest tworzenie oprogramowania zapewniającego wyraźną przewagę konkurencyjną.

ISBN 978-83-283-9552-7