



Software Craftsman

Profesjonalizm, czysty kod i techniczna perfekcja

Tytuł oryginału: The Software Craftsman: Professionalism, Pragmatism, Pride

Tłumaczenie: Zbigniew Waśko

ISBN: 978-83-283-2135-9

Authorized translation from the English language edition, entitled: THE SOFTWARE CRAFTSMAN: PROFESSIONALISM, PRAGMATISM, PRIDE; ISBN 0134052501; by Sandro Mancuso; published by Pearson Education, Inc, publishing as Prentice Hall.
Copyright © 2015 by 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 SA., Copyright © 2016.

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

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

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

Wydawnictwo HELION

ul. Kościuszki 1c, 44-100 GLIWICE

tel. 32 231 22 19, 32 230 98 63

e-mail: helion@helion.pl

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

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

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

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

SPIS TREŚCI

Słowo wstępne	11
Przedmowa	13
Podziękowania	19
O autorze	23
Część I Ideologia i podejście	25
Rozdział 1. Tworzenie oprogramowania w XXI wieku	27
Starszeństwo	29
Nowa rzeczywistość	30
Rozdział 2. Agile, czyli zwinność	33
Dyscypliny Agile ukierunkowane na proces	34
Dyscypliny ukierunkowane na technikę	34
Na czym polega bycie zwinnym?	34
Przełom	35
Poszerzanie kompetencji	35
Ewolucja profesjonalizmu	36
Manifest Agile	36
Zasady uzupełniające Manifest Agile	36
Czas wdrażania zasad Agile	37
Kac agilowy	38
Transformacja częściowa	39
Agile coaching	41

Odrzucanie praktyk o charakterze technicznym	42
Naiwne podejście do tworzenia oprogramowania	42
Są też i dobre wieści	43
Zwinność a rzemiosło programowania	44
Podsumowanie	44
Rozdział 3. Software Craftmanship	47
Lepsza metafora	47
Co mówi Wikipedia?	48
Definicja osobista	48
Definicja krótka	48
Wykraczając poza definicje	48
Rzemiosło, zawód, inżynieria, nauka czy sztuka	49
Historia idei Software Craftmanship	49
Software Craftmanship — spotkanie na szczycie	50
Przekraczanie granic	51
Wymiana rzemieślników	52
Społeczności programistów-rzemieślników	53
Manifest Software Craftmanship	53
Manifest	54
Nie tylko oprogramowanie działające, ale również dobrze wykonane	55
Nie tylko reagowanie na zmiany, ale również ciągłe dodawanie wartości	56
Nie tylko ludzie i interakcje, ale również społeczność profesjonalistów	57
Nie tylko współpraca z klientami, ale również efektywne partnerstwo	58
Lecz niektórzy klienci nie są przygotowani do partnerstwa... ..	59
Problem z manifestem	60
Podsumowanie	61
Rozdział 4. Podejście rzemieślnicze	63
Kto rządzi Twoją karierą?	64
Relacja pracodawca-pracownik	65
Nadążanie za postępem	66
Książki, dużo książek	66
Blogi	68
Techniczne witryny internetowe	69
Za kim podążać?	69
Media społecznościowe	69
Praktyka, praktyka, praktyka	70
Kata	71
Projekty własne	71
Projekty otwarte	73
Programowanie w parach	74
Społeczność	75
Wiedzieć, czego się nie wie	75

Nie samą pracą człowiek żyje	77
Znajdowanie czasu	77
Koncentracja — technika pomodoro	79
Równowaga	79
Podsumowanie	80
Rozdział 5. Bohaterowie, życzliwość i profesjonalizm	81
Nauka mówienia „NIE”	84
Kłęska	84
Wnioski	86
Profesjonalizm	87
Wysuwanie propozycji	89
Opcja nieoczekiwana i realna	90
Oświecone kierownictwo	92
Podsumowanie	93
Rozdział 6. Oprogramowanie działające	95
Działanie oprogramowania to jeszcze nie wszystko	96
Pielęgnowanie ogrodu	97
Ukryte zagrożenie	97
Zakładnicy własnego oprogramowania	98
Zatrudnianie prawdziwych rzemieślników, a nie przeciętniaków	99
Niewłaściwe poczucie czasu	99
Historia długu technicznego	99
Zpracowany zespół	100
Karta zadaniowa testu jednostkowego	103
Mądre gospodarowanie czasem	104
Kod zastany	105
Zmiana nastawienia	106
Własna satysfakcja i zadowolenie klienta	107
Podsumowanie	107
Rozdział 7. Praktyki techniczne	109
Właściwy produkt kontra właściwe wykonanie	109
Kontekst	110
Historia programowania ekstremalnego	111
Praktyki i wartości	113
Dodawanie wartości przez praktykę	114
Odpowiedzialność	119
Pragmatyzm	120
Podsumowanie	121
Rozdział 8. Długa droga	123
Opowieść brazylijskiego nastolatka	123
Koncentracja i wytrwałość	125
A co, jeśli nie wiemy, dokąd zmierzamy?	126

Praca jako inwestycja	127
Samodzielność, mistrzostwo i cel	128
Kariera w ramach firmy	129
Podsumowanie	131

Część II Pełna transformacja 133

Rozdział 9. Rekrutacja 135

Typowy opis stanowiska pracy	135
Zbyt zajęci, by przeprowadzić rozmowę kwalifikacyjną	138
Bez opisów stanowiska pracy	139
A jeśli opis stanowiska pracy jest konieczny?	140
Praca to nie tylko zajęcie	145
Rekomendacje	145
Współpraca ze środowiskiem	146
Definiowanie skutecznych kryteriów wyboru	147
Rekrutacja proaktywna	149
Podsumowanie	150

Rozdział 10. Rozmowa kwalifikacyjna z programistą-rzemieślnikiem 153

Negocjacje biznesowe	154
Rozpoznawanie efektywnego partnerstwa	154
Perspektywa firmy	155
Perspektywa kandydata	156
Dobra rozmowa kwalifikacyjna	158
Właściwy cel	159
Mapa myśli rozmowy kwalifikacyjnej	159
Programowanie w parach jako forma rozmowy kwalifikacyjnej	160
Rozmowy kwalifikacyjne dostosowane do konkretnych potrzeb	163
Sztuka wyboru	164
Przyjmowanie do zespołu istniejącego	
a przyjmowanie do zespołu zupełnie nowego	164
Ćwiczenia w kodowaniu przed rozmową kwalifikacyjną	165
Każdy powinien umieć przeprowadzać rozmowy kwalifikacyjne	166
Z developerami powinni rozmawiać developerzy	167
Podsumowanie	167

Rozdział 11. Antywzorce rozmowy kwalifikacyjnej 169

Nie wymądrzaj się	169
Nie dawaj łamigłówek do rozwiązania	170
Nie zadawaj pytań, na które nie znasz odpowiedzi	170
Nie próbuj udowodniać kandydatowi, że jest głupcem	171
Nie blokuj internetu	172
Nie każ pisać kodu na papierze	172

Nie stosuj algorytmów	172
Nie przeprowadzaj telefonicznych rozmów kwalifikacyjnych	173
Podsumowanie	174
Rozdział 12. Koszt niskiego morale	175
Kac agilowy — niskie morale	175
Koszty zatrudniania deweloperów pracujących od 9.00 do 17.00	177
Ograniczenia wynikające z braku motywacji	180
Zaszczepianie pasji	180
Podsumowanie	182
Rozdział 13. Kult uczenia się	183
Zła motywacja	184
Tworzenie kultu uczenia się	185
Założ klub książki	186
Organizuj techniczne spotkania w porze lunchu	186
Organizuj dyskusje grupowe (przy okrągłym stole)	187
Zmieniaj projekty po każdej iteracji	187
Przeprowadzaj grupowe przeglądy kodu	189
Organizuj sesje ćwiczeniowe	189
Zainicjuj wewnętrzną społeczność praktyków	191
Zorganizuj czas na realizację własnych projektów	191
Nawiąż współpracę z zewnętrznymi społecznościami	192
A jeśli towarzystwo jest odporne?	192
Bądź przykładem	192
Skup się na tych, którym zależy	193
Nie zmuszaj	193
Nie próbuj zmieniać wszystkich	193
Nie uzgadniaj terminów	193
Nie proś o upoważnienie	194
Nie komplikuj	194
Ustal rytm swojego działania	195
Podsumowanie	195
Rozdział 14. Wprowadzanie zmian technicznych	197
Rodzaje sceptycyzmu	197
Bądź przygotowany	201
Od czego zacząć?	202
Zdobądź zaufanie	202
Nabierz doświadczenia	203
Zachęcaj własnym przykładem	203
Nie walcz na wielu frontach	204
Przeprowadzaj iteracje, inspekcje i wdrożenia	205
Strach i niekompetencja	206
Jak przekonać kierownika?	207
Jak przekonać zespół do stosowania metodyki TDD?	208

Jak przekonać sceptyków?	209
Oderwany od świata architekt	210
Pokrzywdzeni	214
Czy naprawdę powinno nas to wszystko obchodzić?	215
Podsumowanie	215
Rozdział 15. Rzemiosło pragmatyczne	217
Jakość jest zawsze oczekiwana	217
Obalanie mitu jakości kosztownej i czasochłonnej	219
Czy musimy wszystko testować?	220
Refaktoryzacja	221
„Jedyny” sposób tworzenia oprogramowania	222
Pomaganie stronie biznesowej	223
Proste i szybkie rozwiązanie	223
W projektach informatycznych nie chodzi o nas	226
Wybitny kontra przeciętny	226
Cztery zasady prostoty	227
Wzorce projektowe	228
Refaktoryzacja do wzorców	228
Rzemiosło a pragmatyzm	230
Podsumowanie	230
Rozdział 16. Kariera programisty-rzemieślnika	233
Bycie rzemieślnikiem	234
Uczciwość i odwaga	235
Rozwój kariery	235
Różne drabiny	236
Drogi i kamienie milowe	237
Budowanie kariery, tylko jedna praca naraz	238
A jeśli nie wiemy, dokąd iść?	240
Różnorodność prac	241
Misja	242
Dodatek A Mity na temat rzemiosła i dalsze wyjaśnienia	243
Deweloper-rzemieślnik a zwykły deweloper	244
Elitaryzm	244
Terminator, czeladnik i mistrz	244
Mistrz rzemiosła	245
Kłapki na oczach	245
Rzemiosło a XP	246
Przywiązanie do praktyk	246
Agile coachowie i kierownicy	246
Nauka zawodu programisty	247
Problem z metaforą	247
Skorowidz	249

5 BOHATEROWIE, ŻYCZLIWOŚĆ I PROFESJONALIZM

W latach 90. ubiegłego wieku zatrudniłem się w dużej międzynarodowej firmie. Po dwóch latach pracy w małych software house'ach i długim procesie rekrutacyjnym wydawało mi się, że wreszcie znalazłem pracę, o jakiej marzyłem. Tworzyliśmy system planowania zasobów przedsiębiorstwa (ERP), podobny do SAP ERP, a naszymi klientami były rządy i wielkie korporacje międzynarodowe. Zespół składał się z bardzo utalentowanych deweloperów, a sam projekt był niezwykle rozbudowany w porównaniu z tymi, nad którymi wcześniej pracowałem. Umiejętności obserwowane u kolegów z zespołu budziły we mnie kompleks niższości. Wszyscy wydawali mi się znacznie lepsi ode mnie, a ja koniecznie chciałem im dorównać. Pragnąłem za wszelką cenę pokazać, że też zasługuję na miejsce w tym zespole. Należałem do zespołu zajmującego się *architekturą* systemu, a więc odpowiedzialnego za rozwijanie najbardziej fundamentalnych składników, infrastruktury komunikacyjnej i warstwy middleware, a także za generowanie kodu (przenoszenie kodu naturalnego do Delphi) i wspieranie zespołów *biznesowych* odpowiedzialnych za funkcje biznesowe systemu oraz interfejs użytkownika.

Projekt był naprawdę potężny. Na wspólnej bazie kodowej pracowało ponad stu deweloperów. Dużo kodu backendowego było tworzone w prawnie zastrzeżonych języku i bazie danych działających w środowisku mainframe, a oprogramowanie pośredniczące (ang. *middleware*) było pisane w C++. Jedyną rzeczą, którą cokolwiek

rozumiałem, było oprogramowanie klienckie pisane w Delphi. Wkrótce też przekonałem się, że nawet moja znajomość Delphi jest, delikatnie mówiąc, niewystarczająca.

Realizacja projektu zaczęła się dziesięć lat wcześniej i gdy przyszedłem, od dwóch lat trwały wyęteżone prace nad przenoszeniem dużej części systemu ze środowiska mainframe na platformę Windows. Zaczynały się też rozmowy nad przeniesieniem systemu do sieci, co było wielkim wyzwaniem dla wszystkich zespołów, ponieważ w większości składały się one z deweloperów pracujących wcześniej w środowisku mainframe, którzy dopiero zaczynali poznawać Delphi. Nie mieli oni też żadnego wzorca, z którym mogliby porównać własną aplikację, gdyż były to pierwszy duży projekt sieciowy w tej branży. Przygotowanie takiej migracji zlecono naszemu zespołowi.

Projekt był realizowany w klasycznym stylu kaskadowym i w ciągu tych dwóch i pół roku, jakie tam spędziłem, nigdy nie rozmawiałem z żadnym z naszych klientów. Żadnego z nich nawet nie widziałem. Sądzę, że ani jeden deweloper nie kontaktował się z klientami. Kierownictwo i jeszcze paru innych ludzi ciągle przydzielali nam nowe zadania i wyznaczali nieprzekraczalne terminy — nie miałem pojęcia, kim byli ci inni ludzie ani co robili. Nie miałem wprawdzie dostępu do prawdziwych liczb, ale z rozmów przy kawie dowiedziałem się, że kontrakty zawierały duże kwoty dotyczące premii i kar. Za każdym razem, gdy klient prosił o nową instalację lub zaktualizowanie oprogramowania, wszyscy wpadali w panikę.

Firma zatrudniała wielu deweloperów (w tym mnie) mieszkających około stu kilometrów od São Paulo (gdzie była siedziba firmy), do którego to miasta byliśmy dowożeni specjalnym autokarem. Musiałem więc wstawać przed 5, aby o 5.30 być już na miejscu zbiórki. Do domu wracałem zazwyczaj około 20, ale gdy zdarzały się nadgodziny, co nie było wcale rzadkością, autokar odjeżdżał beze mnie — musiałem wtedy korzystać z transportu publicznego i docierałem do domu już po północy.

Było coraz gorzej. Wyznaczono nam termin dostarczenia jednego z modułów systemu rządowi Izraela. Ze względu na różnicę stref czasowych i poziom desperacji, jaka w firmie zapanowała, musieliśmy pracować do późna. Skoro nie mogłem korzystać z autokaru, postanowiłem dojeżdżać codziennie samochodem, co zajmowało mi 90 minut w jedną stronę. Firma nie zwracała mi ani za paliwo, ani za opłaty drogowe, ponieważ zapewniała mi darmowy autokar. W ciągu dwóch miesięcy wielokrotnie pracowaliśmy przez 24 godziny bez większych przerw. Po takiej „dniówce” spędzenie 90 minut za kierownicą było zbyt niebezpieczne, więc spałem wtedy w samochodzie stojącym na firmowym parkingu. Trzy albo cztery razy pracowaliśmy nieprzerwanie przez 36 godzin, po czym kładliśmy się w samochodach na czas od 2 do 5 godzin, by potem znów siadać przy biurkach. Gdy koledzy chcieli mnie o coś zapytać, budzili mnie, stukając w szybę samochodu. Oczywiście nigdy nie otrzymaliśmy żadnej zapłaty za nadgodziny. W końcu po wielu rozmowach z zarządem zdecydowano się na opłacenie nam hotelu w pobliżu firmy.

Koledzy z zespołów biznesowych mówili, że to, co robimy, jest głupie — pracujemy na okrągło, śpimy w samochodach, a nikt nam nie płaci za nadgodziny. „Co wy wyprawiacie? Dlaczego się na coś takiego godzicie?” — pytali. Ja jednak nie widziałem w tym nic złego. Byłem samotny, żyłem na własną rękę i kochałem swoją pracę. Nade wszystko chciałem pokazać, na co mnie stać i że jesteśmy w stanie uratować projekt. Chcieliśmy być bohaterami. Dla mnie liczyło się tylko to, że dużo się uczę, pracuję z doświadczonymi deweloperami i robię to, co naprawdę kocham — piszę dobrej jakości kod. Koledzy z zespołu czuli podobnie. Byliśmy wyczerpani, ale w głębi ducha czuliśmy zadowolenie.

Gdy dzisiaj wspominam tamte czasy, widzę, jak szalone to było. Nasze postępowanie było kompletnie nieprofesjonalne. Nigdy nie pytaliśmy, dlaczego mamy robić to czy tamto. Nigdy nie próbowaliśmy dociekać, o co tak naprawdę chodzi klientowi, aby ewentualnie podpowiedzieć mu jakieś alternatywne rozwiązanie. Nie mieliśmy kontaktu z klientami i nie kwestionowaliśmy ich wymagań. Nigdy też nie mówiliśmy, że coś jest niemożliwe. Po prostu wykonywaliśmy swoją pracę, ponieważ wydawało nam się, że jesteśmy profesjonalistami.

W głębi duszy liczyliśmy także na odrobinę uznania i sławy. Marzyło nam się, że będziemy postrzegani jako ci, którzy uratowali projekt, a także jako ci, którzy dokonali rzeczy niemożliwej. Ostatecznie okazało się jednak, że cały nasz wysiłek był niepotrzebny. Termin został przesunięty i nikogo nie obchodziło, że tak ciężko pracowaliśmy. Nie staliśmy się sławni. Nadal byliśmy grupą kiepsko opłacanych deweloperów. Zdołaliśmy na czas wykonać to, co nam zlecono, ale z jakichś, nigdy niewyjaśnionych powodów termin został przesunięty. Sprzedawcy i ludzie odpowiedzialni za stronę biznesową przez cały ten czas pracowali w normalnym wymiarze, a czas wolny spędzali w domach z rodzinami. Tylko my, deweloperzy, pracowaliśmy jak osły, aby zadowolić ludzi, których nawet nie znaliśmy. Przy odgórnym zarządzaniu i systemie nakazowo-kontrolnym większość decyzji po prostu do nas nie docierała. Byliśmy jak robotnicy w fabryce. A jednak nas to cieszyło. Tworzyliśmy naprawdę fantastyczny zespół, a praca wśród wielu utalentowanych i rozumiejących się ludzi sprawiała nam wiele przyjemności.

Dzisiaj wiem, że to, co się wtedy działo, było złe. Boli mnie, gdy sobie przypominam, jak nas traktowano. Boli mnie, gdy sobie uświadamiam, że takie sytuacje są wciąż na porządku dziennym, szczególnie w krajach rozwijających się. Czuję niesmak na myśl o naszym nieprofesjonalnym działaniu — nigdy nie dociekaliśmy, dlaczego mamy robić to czy tamto ani na czym polega prawdziwy problem. Nigdy nawet nie próbowaliśmy szukać alternatywnych rozwiązań. Pozwalaliśmy na to, by nas traktowano jak robotników w fabryce. Zachowywaliśmy się jak robotnicy i to nas cieszyło. A tak naprawdę wyrządzaliśmy krzywdę sobie i firmie. Podczas tych długich godzin harówki popełnialiśmy mnóstwo błędów, włącznie ze skasowaniem o 3 nad ranem całej produkcyjnej bazy danych. Tworzyliśmy rzeczy zupełnie niepotrzebne. Nie wiedzieliśmy,

na czym polegają prawdziwe problemy, i nie stwarzaliśmy sytuacji, które mogłyby nas naprowadzić na właściwe rozwiązania. Pozwoliliśmy na zabranie sobie życia prywatnego, a nawet pogardziliśmy tymi, którzy nie spali w samochodach lub pracowali *tylko* 12 godzin dziennie. Nie byliśmy profesjonalistami, ponieważ nigdy nie powiedzieliśmy „NIE”.

NAUKA MÓWIENIA „NIE”

Bardzo często kierownictwo narzuca nam sztywne terminy i mocno naciska, byśmy ich dotrzyмали. Często są to terminy nierealne, a kierownicy w sposób agresywny starają się wymusić na deweloperach wykonanie zadania na czas. Deweloperzy, zwłaszcza ci młodzi, z reguły ulegają presji i mówią, że zrobią wszystko w wyznaczonym terminie, gdyż boją się sprzeciwić kierownictwu. Mimo że w głębi ducha wiedzą, iż wykonanie zadania w tak krótkim czasie jest prawie niemożliwe, zgadzają się i podejmują zobowiązanie.

Rezultaty są zazwyczaj opłakane. Systemy wdrażane do produkcji są pełne błędów, klienci są niezadowoleni, a firma traci w ich oczach zaufanie. Nawet najcięższa praca po kilkanaście godzin dziennie kosztem czasu, który można by spędzić z najbliższymi, nie zagwarantuje, że system będzie działał poprawnie. A ten sam kierownik, który przydzielił deweloperom zadanie praktycznie niewykonalne, podczas gdy sam spędził weekend w domu z rodziną, będzie im zarzucał brak kompetencji. „Dlaczego u licha przerwaliście walidację na stronie rejestracji? Dlaczego system nie działa prawidłowo? Jak to możliwe, że system nie jest w stanie przetworzyć więcej niż 10 000 transakcji na minutę? Jak można było zapomnieć o tym skrajnym przypadku?”

KLĘSKA

Pracowałem kiedyś w dużej firmie telekomunikacyjnej. Zespołowi marketingowemu spieszyło się z otwarciem nowego portalu mobilnego, nad którym właśnie pracowaliśmy. Chciano go uruchomić w trzech krajach jednocześnie, a zapewniano nas, że w pierwszych dniach tylko niewielka część klientów będzie używała nowej aplikacji. Łączna liczba klientów w tych trzech krajach wynosiła prawie 20 milionów, więc — mówiąc o niewielkiej części — miano na myśli kilkaset tysięcy osób. My takiej wiedzy nie posiadaliśmy. Kierownik przyszedł do nas i powiedział, że chce, aby aplikacja była gotowa przed takim a takim dniem. Chociaż mieliśmy przed sobą kilka miesięcy, byliśmy niemal pewni, że nie zdążymy zrobić wszystkiego, czego żądał. Na jednym z pierwszych spotkań zasygnalizowaliśmy swoje wątpliwości, ale w odpowiedzi usłyszeliśmy tylko, że zadanie musimy wykonać i że jeśli weźmiemy się solidnie do pracy, to na pewno damy radę. Gdy jeszcze próbowaliśmy oponować, dodał, że nie ma innej opcji, więc nie pozostało nam nic innego, jak powiedzieć, że *zrobimy wszystko, co w naszej mocy*.

Tydzień po tygodniu kierownik i analityk biznesowy poszerzali aplikację o kolejne funkcje, ale terminu zakończenia prac nie przesuwali. Wielokrotnie mówiliśmy im, że ledwie dajemy sobie radę z istniejącymi już wymaganiami, więc z nowymi na pewno nie zdążymy. Wracali jednak i stwierdzali, że zespół marketingowy rozpoczął już kampanię reklamową we wszystkich trzech krajach i informacje o nowych funkcjach zostały już podane.

Twierdziliśmy, że zespół marketingowy powinien najpierw porozmawiać z nami i wspólnie powinniśmy ustalić, które funkcje rzeczywiście muszą być zaimplementowane w całości, a które można by zaimplementować tylko częściowo. Mieliśmy kilka pomysłów, dzięki którym niektóre funkcje udałoby się zdecydowanie zmniejszyć albo nawet wyeliminować. Kierownik odparł, że to on jest naszą skrzynką kontaktową i zna wszystkie wymagania marketingowców, więc nie ma potrzeby, byśmy z nimi rozmawiali. Za każdym razem powtarzał: „Niestety, nie ma miejsca na negocjacje. Musicie zaimplementować wszystko”. Każda rozmowa z nim kończyła się tym, że nie mamy wyjścia i musimy pracować tak ciężko, jak tylko potrafimy. Co jeszcze mogliśmy zrobić?

Przez kilka tygodni pracowaliśmy również po godzinach i w weekendy. Oczywiście mam na myśli tylko deweloperów. Kierownik z analitykiem biznesowym znikali zaraz po 18, a w weekendy nawet nie odbierali telefonów. Od samego początku mówiliśmy, że trzeba przygotować określoną infrastrukturę dla prac nad tym projektem, że musimy mieć kwaziprodukcyjne środowisko do testowania aplikacji. Zamiast uzyskać zgodę na wykonanie czegoś takiego, słyszeliśmy, że mamy się skupić na dopracowaniu funkcji biznesowych, ponieważ zespół marketingowy chce opublikować wersję demonstracyjną aplikacji. Na niecały miesiąc przed ostatecznym terminem oznajmiliśmy, że musimy przerwać prace nad funkcjami biznesowymi i zająć się technicznymi aspektami całości, aby zagwarantować prawidłowość działania aplikacji, i w związku z tym nie zdążymy opracować niektórych funkcji. Odpowiedź znów brzmiała *nie*. Mieliśmy skończyć wszystkie funkcje biznesowe. „Nie kombinujcie, to już końcówka, wiem, że dacie radę” — usłyszeliśmy od kierownika. Jeszcze raz poprosiliśmy o możliwość porozmawiania z marketingowcami, z którymi nie spotkaliśmy się od momentu rozpoczęcia prac nad projektem, lecz i tym razem odpowiedź była negatywna. W obliczu zbliżającego się terminu zakończenia prac i ze świadomością, ile jeszcze pracy zostało do wykonania, mogliśmy odpowiedzieć tylko: „Oczywiście, postaramy się”.

W końcu nadszedł ten dzień. Był poniedziałkowy poranek, gdy aplikacja ujrzała światło dzienne. Prace produkcyjne zakończyły się zaledwie kilka godzin wcześniej. Byliśmy zmęczeni i chcieliśmy, żeby to się już skończyło, ale z drugiej strony byliśmy niezwykle dumni z siebie, gdyż wierzyliśmy, że dzięki nam projekt został uratowany. Czekaliśmy na słowa uznania. Czuliśmy się jak bohaterowie. W środku nocy, podczas przerwy na kawę, rozmawialiśmy nawet o czekających nas podwyżkach i awansach.

System wystartował, ale po niecałej godzinie już nie działał. Obciążenie było zbyt duże. Po trzech godzinach udało nam się przywrócić go do życia, ale znów tylko na parę godzin. Sytuacja powtórzyła się jeszcze wiele razy w następnych pięciu dniach. Wszyscy, włącznie z marketingowcami (wreszcie ich zobaczyliśmy), zaglądaliśmy sobie przez ramię, pytając, co się dzieje z tym systemem. Testy? Nie. Było tylko kilka. Kto miał czas na pisanie testów? Mimo że byliśmy małym zespołem, stworzyliśmy kod niezwykle chaotyczny, w którym wprowadzanie zmian było niezwykle trudne, podobnie zresztą jak testowanie — niemal wszystko trzeba było sprawdzać ręcznie.

Podsumowując, firma została zasypana skargami i pretensjami klientów. Po kilku tygodniach udało nam się pokonać wszystkie problemy i ustabilizować system, ale oczywiście zamiast premii i awansów zyskaliśmy tylko złą reputację. Kierownik tłumaczył się przed zespołem marketingowym, że to my, deweloperzy, twierdziliśmy, że wszystko da się zrobić, a on polegał na naszej opinii i wierzył w nasze umiejętności. Przecież on nie pisał kodu, prawda? Więc to nie jego wina.

Gdy już było po wszystkim, marketingowcy powiedzieli nam, że gdyby wiedzieli o naszych kłopotach i wątpliwościach co do wydolności systemu, zrezygnowaliby z kilku funkcji. Zgłaszali zapotrzebowanie na coraz więcej rzeczy, ponieważ sądzili, że jesteśmy w stanie je zaimplementować i że wszystkie pozafunkcjonalne wymagania mamy pod kontrolą. Nikt im nie powiedział, że jest inaczej.

Ostatecznie większość z nas rozstała się z firmą, co zmusiło jej zarząd do szybkiego zatrudnienia nowych deweloperów. Firma nie miała czasu na wybranie najlepszych kandydatów, więc szybko wpadła w jeszcze większe tarapaty, gdyż nowi deweloperzy nie mieli pojęcia o tych wszystkich pułapkach, jakie na nich czyhały w pozostawionym przez nas kodzie. Zapowiedziano wypuszczenie drugiej wersji aplikacji w ciągu trzech miesięcy, ale z tego, co słyszałem, udało im się to dopiero po trzech kwartałach.

WNIOSKI

W powyższej historii od razu widać, że kierownik nie działał profesjonalnie. Nie dbał o zespół i myślał tylko o własnym awansie, podczas gdy całą pracę wykonywali za niego inni. Aż ciśnie się na usta, że to on zawinił i on powinien być zwolniony oraz że deweloperzy powinni się bronić i opowiedzieć wszystkim, jak się kierownik zachowywał i jak oni ciężko pracowali, aby ratować projekt. Owszem, możemy to wszystko powiedzieć, ale to już niczego nie zmieni. Teraz jest już za późno.

Całej tej sytuacji można było uniknąć, a winą za niepowodzenie powinno się obarczyć nas, deweloperów. My też nie działaliśmy profesjonalnie. Nie powinniśmy mówić *spróbujemy*, jeśli wiedzieliśmy, że zadanie jest niewykonalne. Nawet jeśli wierzyliśmy, że możliwe jest zaimplementowanie wszystkich funkcji, powinniśmy zdawać sobie

sprawę, że nie zdołamy wszystkiego przetestować, a więc nie będzie żadnej pewności co do sprawności aplikacji. Niezależnie od niechęci do przeciwstawiania się kierownikowi było w nas coś, co nakazywało nam godzić się na jego warunki. Gdzieś w głębi odczuwaliśmy pragnienie pokazania światu, jacy jesteśmy dobrzy. Mimo że wiedzieliśmy, jak trudne to będzie, to jednak cały czas liczyliśmy, że jest niewielka szansa na sukces, a jeśli się uda, będziemy bohaterami. Będą o nas mówić jako o tych, dzięki którym udała się wielka kampania reklamowa. Gdzieś w głębi liczyliśmy, że tak się stanie.

W tej sytuacji nie powinniśmy nigdy mówić, że spróbujemy, ani myśleć o bohaterstwie. Od początku powinniśmy naciskać na kierownika w sprawie kontaktów z działem marketingu. Powinniśmy się zorientować, że kierownik nie mówi wszystkiego marketingowcom, skoro ciągle dodaje nam nowe funkcje, wiedząc, że nasze możliwości są ograniczone. Powinniśmy dokumentować przebieg każdego spotkania, podczas którego informowaliśmy kierownika o istniejących ograniczeniach i potrzebie przygotowania systemu do obsłużenia dużej liczby klientów. Kierownik, choć robił to fatalnie, wykonywał swoją pracę — pilnował, aby aplikacja zawierała wszystkie funkcje, jakich żądali marketingowcy. My powinniśmy pracować lepiej i wykazując więcej realizmu, nie przyjmować na siebie odpowiedzialności za wykonanie czegoś, co — o czym wiedzieliśmy — będzie kiepskiej jakości i nie będzie się nadawało do wdrożenia. Gdy zaczęto dodawać nowe funkcje, powinniśmy stanowczo zażądać spotkania z zespołem marketingowym. Powinniśmy wyraźnie powiedzieć, że system pełen nawet najlepszych funkcji będzie zupełnie bezużyteczny, jeśli będzie działał źle lub wcale. Myślenie o bohaterstwie sprawiło, że nie byliśmy profesjonalistami. Nie umieliśmy powiedzieć *nie*.

PROFESJONALIZM

Trudne do dotrzymania terminy zdarzają się nam bardzo często. Najlepszym sposobem radzenia sobie z nimi jest przeanalizowanie wszystkiego, co trzeba zrobić, oraz poinformowanie o wszelkich zagrożeniach, jakie dostrzegamy, a także o obawach, jakie mamy. Wszystkie obawy i niejasności należy zgłaszać najszybciej jak to możliwe. Powinniśmy też określić, co jesteśmy w stanie wykonać należyście przed wyznaczonym terminem. Przez wykonanie należyte rozumiem napisanie kodu i wszechstronne przetestowanie go w środowisku podobnym do produkcyjnego.

Kierownicy często nie traktują poważnie słów deweloperów, gdy ci mówią, że nie są w stanie wykonać należyście wszystkiego przed wyznaczonym terminem. Zaczynają się wtedy negocjacje, ale niemający umiejętności negocjacyjnych i dobrze przygotowanych argumentów deweloperzy najczęściej ulegają presji i biorą na swoje barki więcej, niż mogą udźwignąć. Niektórzy kierownicy bywają bardzo przekonujący w namawianiu do przyjęcia terminu, z którego trudno się wywiązać. „Chłopaki, wiecie, jakie to jest ważne. Zdaję sobie sprawę, że nie będzie łatwo, ale wierzę w was. To jest naprawdę

potrzebne. Jestem pewien, że jeśli się przyłożycie, to dacie radę”. Aby uniknąć konfrontacji lub zademonstrować ochotę do pracy, deweloperzy odpowiadają: „OK. Damy z siebie wszystko”.

Gdy mówimy, że spróbujemy, zwykle druga strona odbiera to jako naszą gotowość do wykonania zadania. Nasze *spróbuję* jest równoznaczne z *tak, będzie zrobione*. Brzmi to również jak wyznanie, że na co dzień nie pracujemy zbyt ciężko. Zupełnie jak byśmy powiedzieli, że mamy spory zasób energii i w każdej chwili możemy go wykorzystać, aby przyspieszyć pracę.

Skoro wiemy, że wykonanie zadania w terminie jest praktycznie niemożliwe, to dlaczego mówimy kierownictwu, że spróbujemy je wykonać? Co to znaczy *spróbujemy*? Czy to znaczy, że jesteśmy w stanie wykonać każde zadanie, jeśli tylko będziemy pracować intensywniej? Czy to znaczy, że zobowiązujemy się pracować po godzinach i w weekendy? Jeśli tak, to czy to coś zmienia? Jeżeli nie, wówczas oznacza to, że albo na co dzień nie pracujemy zbyt intensywnie, albo po prostu kłamiemy.

W swojej książce *The Passionate Programmer* Chad Fowler twierdzi, że mówienie *tak* tylko dlatego, by nie zawieść drugiej strony, jest zwykłym kłamstwem. Idzie nawet dalej, utrzymując, że mówienie *tak* jest nawykiem uzależniającym i wyniszczającym. Jest to zły nawyk, który tylko pozornie wygląda na dobry.

Z reguły nie lubimy mówić *nie*. To źle brzmi. Gdy mówimy *nie*, czujemy się jak przegrani. Wydaje nam się, że jesteśmy gorsi od innych i rozczarowaliśmy kolegów z zespołu. Chcemy odnieść sukces, ale musi to być pewny strzał. Chociaż wydaje się to odruchem pozytywnym, jest też przejawem egoizmu. Należy pamiętać, że gdy mówimy *tak*, inni mogą na tej podstawie coś zaplanować. Nasi przełożeni mogą przedstawić plany swoim przełożonym, innym zespołom, klientom lub partnerom biznesowym. Wypowiadanie się w sposób nierzetelny i niejednoznaczny może wyrządzić wiele szkód całej firmie. Profesjonalizm oznacza bycie uczciwym wobec siebie, swojego zespołu, kierownictwa i klientów.

Czasami czujemy, że nasz kierownik nie mówi swojemu szefowi — lub komukolwiek, kto finansuje projekt — wszystkiego o bieżącym stanie projektu. Wyczuwamy, że nie informuje innych ludzi o naszych obawach, lecz zawsze mówi im, iż wszystko jest w porządku. Powinniśmy go wtedy spytać, dlaczego tak robi. Jeśli sytuacja jest naprawdę zła, powinniśmy go uprzedzić, że rozdmuchamy sprawę i gdy o wszystkim dowie się jego szef, poprosimy o spotkanie w celu przedyskutowania kroków zaradczych. Może brzmi to groźnie i agresywnie, ale będzie z korzyścią dla zespołu, firmy i nierzadko także dla naszego kierownika. Sztuki negocjowania powinniśmy się uczyć wszyscy. Tak jak kierownictwo wykonuje swoją pracę, my powinniśmy wykonywać swoją — podejmując trudne decyzje i nie unikając konfrontacji. Jeśli robimy to w sposób uczciwy i transparentny, jest duża szansa, że nikt nie ucierpi, a zespół i firma zyskają.

WYSUWANIE PROPOZYCJI

Mówienie zawsze *nie* też nie jest podejściem profesjonalnym. Po każdym *nie* powinna następować lista rozwiązań alternatywnych. Zanim to słowo wypowiemy, powinniśmy przeanalizować problem i zaproponować jakieś wyjście z sytuacji. Nie zawsze takowe znajdziemy, ale przynajmniej powinniśmy próbować. Czasami nawet niedopracowany pomysł może pomóc w znalezieniu dobrego rozwiązania. Ostatecznie wszyscy powinniśmy uczestniczyć w rozwiązywaniu problemów i powinno być wybrane rozwiązanie najlepsze, bez względu na to, kto je zaproponował.

Mówienie *nie* bez proponowania czegokolwiek w zamian w niczym nie pomaga, ponieważ druga strona niewiele może z taką naszą odpowiedzią zrobić. Pamiętam zasłyszaną kiedyś opowieść o dyrektorze, który właśnie objął stanowisko w nowym dla siebie mieście i poprosił dwie asystentki, aby mu kupiły sok pomarańczowy. Pierwsza pobiegła do sklepu za rogiem, zapytała o sok i po 5 minutach wróciła z informacją, że w sklepie nie mają ani soku, ani pomarańczy. Druga wróciła dopiero po 20 minutach i zanim zdążyła cokolwiek powiedzieć, dyrektor zapytał: „Już wiem, że w sklepie za rogiem nie ma soku pomarańczowego, dlaczego więc tak długo cię nie było?”. „Tak” — odparła. „W sklepie nie mają soku pomarańczowego, ale mają ananasowy i jabłkowy. Ananasowy jest świeżo wyciskany, a jabłkowy mają butelkowany. Dowiedziałam się również, że trochę dalej jest supermarket, gdzie można kupić pomarańcze. W ciągu pięciu minut może pan mieć sok ananasowy lub jabłkowy albo w ciągu pół godziny zrobię świeży sok pomarańczowy. Muszę tylko wiedzieć, co pan wybiera”.

Pierwsza asystentka, chociaż wróciła szybko, pozostawiła swojego szefa z niczym. Musiał sam rozwiązywać problem, nie znając w ogóle okolicy. Druga asystentka zaproponowała mu kilka opcji, z których mógł wybrać według niego najlepszą. Dobre chęci zawsze się liczą, nawet więc gdy mówimy *nie*, zawsze powinniśmy dążyć do tego, by powiedzieć *tak*.

Czasami są jednak problemy, których po prostu nie umiemy rozwiązać. W takich sytuacjach powinniśmy się do tego uczciwie przyznać, i to najszybciej, jak się tylko da. Należy też okazać gotowość do przyjrzenia się problemowi i dołożenia wszelkich starań w celu jego rozwiązania, ale z wyraźnym zaznaczeniem, że nie obiecujemy nic więcej poza dzieleniem się swoją wiedzą. Najlepsze, co możemy zrobić, to jak najczęściej informować o swoich postępkach; umożliwi to zespołowi podejmowanie decyzji, co robić z każdą nową porcją informacji. Dzięki nowej informacji zespół może rozwinąć inne pomysły, a także zaoferować nam pomoc w obszarach, w których nie czujemy się zbyt pewnie. Takie zachowanie sprawi, że nikt w zespole nie będzie wątpił w nasze słowa, gdy stwierdzimy, że coś wiemy. Bycie uczciwym nawet w najtrudniejszej sytuacji jest oznaką profesjonalizmu.

OPCJA NIEOCZEKIWANA I REALNA

Całkiem niedawno pracowałem nad projektem dla banku inwestycyjnego. Projekt dotyczył raportowania transakcji dla instytucji nadzoru bankowego. Termin zakończenia prac został narzucony przez wspomnianą instytucję, więc w banku nie było z kim rozmawiać na ten temat. Sankcją za niedotrzymanie terminu mogła być wysoka kara pieniężna lub zakaz sprzedaży pewnych produktów, co oznaczałoby poważne naruszenie reputacji banku.

Mieliśmy bardzo zdolny zespół i pracowaliśmy ciężko, żeby zdążyć. Nadszedł jednak moment, w którym stało się jasne, że nie potrafimy spełnić wszystkich obowiązkowych wymagań — mieliśmy problem z automatyzacją procesu raportowania wszystkich typów transakcji wymaganych przez nadzór.

Kierownictwo naciskało, żebyśmy pracowali po godzinach i w weekendy, gdyż stawka była naprawdę wielka. Wszystkie banki inwestycyjne w Wielkiej Brytanii przechodziły przez to samo, a o nowych przepisach mówiło się we wszystkich wiadomościach finansowych. Zarząd wciąż powtarzał, że negocjowanie warunków projektu lub terminu nie wchodzi w grę, gdyż nikt nie może w tej sprawie nic zrobić. Albo raportowanie będzie działało, albo zapłacimy kary i stracimy dobrą reputację.

W wyniku wymagań instytucji nadzorującej wyznaczano różne terminy wprowadzania raportowania transakcji spełniających określone kryteria. Było jasne, że nasza aplikacja będzie się rozwijać, a zatem nie mogliśmy sobie pozwolić na rezygnację z przyjętych standardów i praktyk kodowania tylko po to, aby zdążyć przed pierwszym terminem. Spowodowałoby to obniżenie jakości naszej bazy kodowej, a to z kolei znacznie utrudniłoby dotrzymanie terminów dla pozostałych typów transakcji.

Zgodziliśmy się na wydłużenie dnia pracy w tygodniu, ale odmówiliśmy pracy w weekendy, a także odrzuciliśmy propozycję zarządu dotyczącą poszerzenia składu zespołu. Uznaliśmy, że sytuacja tylko by się przez to pogorszyła. Nadal składaliśmy zarządowi codzienny raport z postępu naszych prac i ciągle pokazywaliśmy, że pierwszy termin jest zagrożony. Wystarczyło zresztą rzucić okiem na naszą tablicę kanban, aby zobaczyć, na jakim etapie jesteśmy. Po dwóch czy trzech tygodniach oznajmiliśmy, że nie będziemy już pracować po godzinach.

Nie muszę chyba mówić, jaką to wywołało panikę i strach przed konsekwencjami niedotrzymania terminu. Wytrwanie we własnym postanowieniu i niepoddanie się presji kierownictwa były dla nas niezwykle trudne. Ale wytrzymaliśmy. Wiedzieliśmy, że obniżenie standardów i pracowanie przez siedem dni w tygodniu spowoduje, że się wypalimy i realizacja projektu jako całości będzie zagrożona.

Na szczęście pracowaliśmy zgodnie z zasadami Agile i na białej tablicy zawsze był widoczny stan naszych prac oraz to wszystko, co nas blokowało lub było wąskim gardłem. Każdy z nas miał też ciągły dostęp do rejestru produktu. Wizualizacje tego, co już było zrobione, i tego, co należało jeszcze zrobić, pomagały zarządowi w orientowaniu się, jaka jest aktualna sytuacja, a nam pomagały w przekonywaniu, że terminu nie da się dotrzymać, nawet jeśli będziemy pracować przez siedem dni w tygodniu.

Powiedzenie *nie* już pierwszego dnia i nieukrywanie niczego przed stroną biznesową daje poczucie wolności. Każdy ma wtedy swobodę w poszukiwaniu innych realnych rozwiązań. Okazało się, że istniały dwa sposoby dostarczania raportów do instytucji nadzorującej. Pierwszy miał polegać na pobieraniu informacji generowanych przez system administracyjny, analizowaniu ich, uzupełnianiu, przekształcaniu i wysyłaniu do instytucji nadzorującej w sposób całkowicie zautomatyzowany — i to właśnie miał robić system, nad którym pracowaliśmy. Drugi sposób to ręczne wysyłanie wygenerowanych przez banki plików CSV za pośrednictwem interfejsu dostarczonego przez instytucję nadzorującą. Ze względu na dużą liczbę transakcji uznano już na początku (jeszcze przed rozpoczęciem prac nad naszym projektem), że drugi sposób jest praktycznie niewykonalny i odrzucono go.

Pracowaliśmy nad pierwszą opcją, ale skoro już wiedzieliśmy, że system nie będzie ukończony na czas dla wszystkich typów transakcji, jakie miały być raportowane, zaczęliśmy analizować wykonalność metody ręcznej. Na początku nikt tego nie brał pod uwagę, ale teraz sytuacja wyglądała inaczej. Raportowanie większości typów transakcji udało nam się już zautomatyzować i na raportowanie ręczne przypadało stosunkowo niewiele pracy. Przekalkulowaliśmy koszty ręcznego przetwarzania i wysyłania raportów przez oddelegowanych (lub wynajętych) do tej pracy ludzi i porównaliśmy je z ceną, jaką przyszłoby zapłacić za niedotrzymanie terminu. Okazało się, że pierwsza opcja jest z biznesowego punktu widzenia znacznie lepsza. Podjęto więc stosowne decyzje i, pomijając kilka drobnych problemów, które jeszcze trzeba było rozwiązać, pierwsze raporty zostały wysłane w terminie.

Gdybyśmy od razu nie powiedzieli *nie* i ukrywali prawdę przed zarządem, licząc, że jakoś nam się uda i zostaniemy bohaterami, prawdopodobnie nasze wysiłki zakończyłyby się niepowodzeniem, a firma poniosłaby wielkie straty. Powiedzenie *nie* uwolniło nas od myślenia wyłącznie o zbliżającym się terminie i pozwoliło skupić się na poszukiwaniu innych rozwiązań zaistniałego problemu.

Opisane wyżej połączenie metod ręcznej i automatycznej, chociaż nie było rozwiązaniem idealnym, to jednak pozwoliło nam na spokojne kontynuowanie prac przy jednoczesnym spełnieniu wszystkich wymagań instytucji nadzorującej.

Zyskaliśmy czas na zautomatyzowanie całego procesu raportowania i wcale nie musieliśmy rezygnować z naszych standardów kodowania. Co więcej, mogliśmy stopniowo uruchamiać kolejne etapy automatyzacji, co pozwalało na dokładne sprawdzenie, czy wszystko działa zgodnie z oczekiwaniami.

OŚWIECONE KIEROWNICTWO

Dobre kierownictwo rozumie, że jest częścią zespołu i musi współpracować z deweloperami w dążeniu do osiągnięcia wspólnego celu. Dobre kierownictwo nie wprowadza podziału na „my” i „oni”. Powinno znać bieżący stan projektu i wspólnie z zespołem ustalać, co będzie możliwe do wykonania w następnym przedziale czasowym.

Kierownictwo powinno doceniać naszą postawę, gdy mówimy, że pewnych rzeczy nie będziemy w stanie wykonać. Chociaż nie jest to dobra informacja, to jednak pozwala uniknąć znacznie większych problemów. Szczerość wzajemnych relacji umożliwia zespołowi i kierownictwu przygotowanie się do trudnych sytuacji. Zapalenie czerwonej lampki najwcześniej, jak to tylko możliwe, w celu zasygnalizowania, że coś jest nie tak, pozwala zespołowi i wszystkim zainteresowanym na zachowanie się w sposób pragmatyczny i podjęcie prób rozwiązania problemu. Ludzie uwolnieni od balastu niewykonalnego zadania stają się bardziej kreatywni i lepiej sobie radzą z problemami. Jeśli wywiązujemy się z powierzonych nam zadań, gdy mówimy *tak*, to kierownictwo będzie dużo bardziej skłonne zaufać nam również wtedy, gdy powiemy *nie*.

Mądrzy ludzie wiedzą, że nie wiedzą wszystkiego i że w grze drużynowej liczą się wszyscy członkowie zespołu. Jak głosi mądrość ludowa: łańcuch jest tak mocny, jak jego najsłabsze ogniwo. Mądre kierownictwo rozumie, że pewne rzeczy mogą być trudne do wykonania i trzeba na to więcej czasu. Docenia, gdy twardo mówimy, w czym rzecz, i jednocześnie sugerujemy inne rozwiązania. Takie podejście pokazuje, że wszyscy tworzą jedną drużynę i dążą do tego samego celu. Mając więcej informacji i kilka propozycji wyjścia z trudnej sytuacji, kierownictwo pewnie podejmuje decyzje; z pewnością też bardziej sobie ceni ludzi, którzy te propozycje zgłaszają.

Dobre kierownictwo jest częścią zespołu. Jest z nim na dobre i na złe. Jeśli wszyscy zostają po godzinach, kierownicy powinni zrobić to samo, choćby po to, by zamówić pizzę, wesprzeć moralnie i od czasu do czasu opowiedzieć coś dowcipnego. Zespół czuje się lepiej, gdy wszyscy jadą na tym samym wózku, znoszą te same niewygody i jednakowo cieszą się z sukcesu. W takim środowisku ludzie chętniej dają z siebie wszystko i bardziej skupiają się na wykonaniu zadania. Dobre kierownictwo chroni deweloperów przed naciskami z zewnątrz, robi wszystko, aby usunąć wszelkie

przeszkody, i stara się stworzyć im jak najlepsze warunki do pracy. Kierownictwo powinno być emanacją harmonii i to do niego należy dbanie o to, by w zespole panowała zdrowa atmosfera i poczucie wspólnoty.

PODSUMOWANIE

Bycie profesjonalistą i dbanie o klientów nie oznacza godzenia się na wszystko, czego żądają. Naszym zadaniem jest pomaganie klientom w ustaleniu, jakie są ich rzeczywiste potrzeby i jak mogą je zaspokoić. Profesjonalista kieruje się własną etyką i swoim kodem postępowania. Prawdziwy profesjonalista nigdy nie zrobi nic, co mogłoby zaszkodzić klientowi, mimo że klient jest gotów za to zapłacić. Klienci rzadko rozumieją specyfikę projektów informatycznych i to my mamy im uświadamiać, jak pewne decyzje mogą wpływać na tego rodzaju projekty. To my musimy powiedzieć *nie*, jeśli klient prosi o coś, co według nas nie zadziała lub czego nie da się wykonać w wyznaczonym czasie. Dokładnie tego samego oczekujemy od dobrego prawnika, księgowego czy lekarza. Nie mówimy tym profesjonalistom, co i jak mają robić. Nawet by nam nie pozwolili. Idziemy do nich z problemem (lub potrzebą) i oczekujemy, że na podstawie własnego doświadczenia powiedzą nam, jakie mamy opcje do wyboru i z jakim ryzykiem każda z nich się wiąże. Oczekujemy, że otrzymamy wystarczająco dużo informacji, aby w pełni świadomie podjąć taką czy inną decyzję. Jeśli jednak spróbujemy skłonić ich do zrobienia czegoś, co uważają za niewłaściwe lub szkodliwe, po prostu powiedzą *nie*. I my też powinniśmy tak postępować.

SKOROWIDZ

A

adaptacyjny rozwój oprogramowania, ASD, 33
Agile, 33
 Alliance, 33
 coach, 246
 coaching, 41
antywzorce rozmowy kwalifikacyjnej, 169–174
API, Application Programming Interface, 27
architektura sytemu, 81
ASD, Adaptive Software Development, 33
atrapa, mock, 161

B

BDUF, 35
błędy, 97
brak motywacji, 180
budowanie kariery, 238

C

cel, 128
ciągła integracja, 117
CV, 148
czeladnik, 244

Ć

ćwiczenia
 kata, 71
 w kodowaniu, 165

D

debugowanie, 103
deweloper-rzemieślnik, 244
deweloperzy, 142, 244
 bierni, 198
 cyniczni, 198
 fanatycy, 201
 irracjonalni, 199
 niedoinformowani, 198
 niepewni, 201
 nieudolni, 200
 obojętni, 199
 oderwani od świata architektki, 200
 pokrzywdzeni, 200
 szefujący, 199
 wypaleni, 198
 zabiegani, 198
dług techniczny, 99
dodawanie wartości, 114
doświadczenie, 203

DSDM, 33
dyscypliny ukierunkowane
 na proces, 34
 na technikę, 34
dyskusje grupowe, 187
działające oprogramowanie, 96
dzielenie się wiedzą, 188

E

efektywne partnerstwo, 154
elitaryzm, 244

F

FDD, Feature-Driven Development, 33

G

grupowe przeglądy kodu, 189

I

IDE, Integrated Development Environment, 15
ignorancja drugiego poziomu, 76
inspekcje, 205
interfejs programistyczny aplikacji, API, 27
inwestycja, 127
iteracje, 187, 205

J

J2EE, 28
jakość, 97, 217, 219
JEE, 101
język, 201
 UML, 28

K

kac agilowy, 175
kamienie milowe, 237
kariera, 64, 126, 129, 233, 238
karta zadaniowa testu, 103
kata, 71
kategorie sceptyków, 198
kierownictwo, 92

kierownik, 246
klapki na oczach, 245
kod zastany, 105
koncentracja, 79, 125
kontekst, 110
koszty zatrudniania deweloperów, 177
kryteria wyboru, 146

L

liczby bezwzględne, 139
listy technologii, 140
LSCC, 17, 21, 23

M

Mancuso Sandro, 23
Manifest Agile, 32, 36
Manifest Software Craftsmanship, 47, 53
mapa myśli rozmowy kwalifikacyjnej, 159
media społecznościowe, 69
metafora, 247
metodyka RUP, 222
middleware, 81
mikrozarządzanie, 95
misja, 242
mistrz, 244
mistrz rzemiosła, 245
mistrzostwo, 128
mity i nieporozumienia, 243
MMF, Minimum Marketable Feature, 110
model
 start-upu oszczędnego, 31
 tworzenia systemów dynamicznych, 33
morale zespołu, 175
motywacje, 184
mówienie nie, 84, 89
myślenie niezależne, 190

N

nauka zawodu programisty, 247
negocjacje biznesowe, 154
niekompetencja, 206
niskie morale, 175

O

obowiązek, 210
 OCP, Open-Closed Principle, 221
 oderwany od świata architekt, 210
 odpowiedzialność, 119, 210
 odwaga, 235
 ogłoszenie, 135, 141
 opcja
 nieoczekiwana, 90
 realna, 90
 opis stanowiska pracy, 135, 140, 142
 oprogramowanie
 działające, 95
 pośredniczące, 81
 overengineering, 116

P

pasja, 180
 PBR, Product Backlog Refinement, 223
 perspektywa
 firmy, 155
 kandydata, 156
 podejście
 naiwne, 42
 rzemieślnicze, 63
 pokrzywdzeni, 214
 pomijanie wewnętrznych awansów, 140
 postęp, 66
 praca, 77, 127, 145
 pragmatyzm, 120, 230
 praktyki, 70
 techniczne, 109, 197
 XP, 113, 114
 proces RUP, 28
 profesjonalizm, 87
 programista-rzemieślnik, 99, 153, 233
 programowanie
 ekstremalne, XP, 33, 111, 246
 sterowane funkcjonalnością, FDD, 33
 sterowane testami, TDD, 34
 w parach, 74, 117, 160
 zwinne, 32

projekty
 otwarte, 73
 własne, 71
 prostota, 201, 227
 przekonywanie
 kierownika, 207
 sceptyków, 209
 zespołu, 208
 przyjmowanie do zespołu, 164

Q

QA, 39, 102

R

realizacja własnych projektów, 191
 refaktoryzacja, 98, 119, 221
 do wzorców, 228
 rekomendacje, 145
 rekrutacja, 135
 proaktywna, 149
 rekruter, 169
 relacja pracodawca-pracownik, 65
 rodzaje sceptycyzmu, 197
 rola, 143
 rozmowa kwalifikacyjna, 153, 158, 163, 166
 rozwiązanie backendowe, 224
 rozwój kariery, 235
 równowaga, 79
 RUP, Rational Unified Process, 28, 222
 rytm działania, 195
 rzemieślnik, 234
 rzemiosło, 243
 pragmatyczne, 217
 programowania, 44

S

samodzielność, 128
 satysfakcja, 107
 sceptycyzm, 197
 Scrum, 33
 sesje ćwiczeniowe, 189
 słowa kluczowe, 139

Software Craftsmanship, 18, 32, 47–60
SOLID, 229
społeczność, 75
stanowisko pracy, 135
stosowanie praktyk XP, 113
strach, 206

T

TDD, Test-Driven Development, 34, 74, 116
techniczne spotkania, 186
technika pomodoro, 79
technologie, 143
terminator, 244
test, 115
 akceptacyjny, 224
 jednostkowy, 103
testowanie, 220
 zautomatyzowane, 115
transformacja pełna, 133
transformacje agilowe, 40
tworzenie
 atrap, 161
 oprogramowania, 222, 241

U

UAT, User Acceptance Testing, 224
uczciwość, 235
uczenie się, 183, 185
UML, Unified Modeling Language, 28
umowa SLA, 30
upoważnienie, 194
usprawnienia, 188

W

walidacja decyzji, 188
wartość biznesowa, 113
wdrażanie zasad Agile, 37

wdrożenia, 205
WIP, Work In Progress, 40
właściwe wykonanie, 109
właściwy produkt, 109
wprowadzanie praktyk technicznych, 197
wspólne uczenie się, 188
współpraca
 z zewnętrznymi społecznościami, 192
 ze środowiskiem, 145
wymagania, 139
 pozafunkcjonalne, 224
wytrwałość, 125
wzorce projektowe, 228

X

XP, Extreme Programming, 33, 111, 246

Z

zadawanie pytań, 170
zadowolenie klienta, 107
zasada otwarte-zamknięte, 221
zaszczepianie pasji, 180
zaufanie, 202
zespół, 100
zintegrowane środowisko programistyczne,
 IDE, 15
zła
 motywacja, 184
 selekcja, 140
złe wymagania, 140
zmuszanie, 193
znajdowanie czasu, 77
zwinność, 44

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



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

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

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

GRUPA WYDAWNICZA

 **Helion SA**

Programuj profesjonalnie — liczą się pragmatyzm, perfekcja i cel!

Coraz więcej mówi się o dobrych praktykach programistycznych, a mimo to wciąż zdarzają się nieudane produkty. Jest wiele przyczyn tego stanu rzeczy, np. niewłaściwe zarządzanie projektami, brak wypracowanych metod rekrutacji specjalistów i kierowania zespołem. Odpowiedzią na te problemy jest wdrożenie rzemieślniczego podejścia do pracy programisty.

Idea ta, znana jako software craftsmanship, kładzie nacisk na profesjonalizm, techniczną perfekcję i zadowolenie klienta. Autor, współzałożyciel największej organizacji deweloperów-rzemieślników, proponuje praktyki programistyczne zaczerpnięte m.in. z metodyk programowania ekstremalnego, zwinnego i odchudzonego. Dzięki tej postawie Twój zespół może osiągnąć najwyższy poziom technicznej doskonałości tworzonych projektów!

Z książki dowiesz się, jak:

- ▶ polepszyć jakość oprogramowania dzięki podejściu rzemieślniczemu
- ▶ utrzymać wysoki poziom pisanego kodu i obsługi klienta
- ▶ być pragmatykiem — a nie dogmatykiem
- ▶ zaszcześcić w zespole prawdziwy kult uczenia się

Sandro Mancuso — ekspert programistyczny z bogatym doświadczeniem w korporacjach, prelegent na konferencjach branżowych. Zwolennik idei programowania mistrzowskiego i ekstremalnego.

W serii znajdziesz też bestsellery Roberta C. Martina:



Helion

44672 numer katalogowy

księgarnia internetowa

<http://helion.pl>

zamówienia telefoniczne

☎ 0 801 339900

☎ 0 601 339900

Informatyka w najlepszym wydaniu

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

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

**PRENTICE
HALL**

ISBN 978-83-283-2135-9



cena: 59,00 zł

sięgnij po WIĘCEJ



KOD KORZYŚCI