O'REILLY®

Debugging Teams

Przez współpracę do lepszej produktywności

Brian Fitzpatrick, Ben Collins-Sussman

Helion

Tytuł oryginału: Debugging Teams: Better Productivity through Collaboration

Tłumaczenie: Piotr Cieślak

ISBN: 978-83-283-4338-2

© 2018 Helion S.A.

Authorized Polish translation of the English edition of Debugging Teams ISBN 9781491932056 © 2016 Ben Collins-Sussman, Brian W. Fitzpatrick

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

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

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w 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 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
h*ttp://helion.pl/user/opinie/debtea*
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

1

Mit genialnego programisty

Ponieważ książka ta jest poświęcona problemom wynikającym ze społecznych aspektów pracy twórczej, logicznie jest skupić się na jedynej zmiennej, nad którą bezsprzecznie masz kontrolę: sobie.

Niedoskonałość jest rzeczą ludzką. Zanim jednak zaczniesz dopatrywać się „bugów" u współpracowników, musisz poznać te, które tkwią w Tobie. Chcemy skłonić Cię do zastanowienia się nad własnymi reakcjami, zachowaniami i postawami — mamy nadzieję, że w zamian zyskasz praktyczny wgląd w to, jak być bardziej efektywnym i spełnionym inżynierem oprogramowania. Będziesz mniej czasu spędzać na zmaganiach z problemami interpersonalnymi, a więcej na pisaniu doskonałego kodu.

Główną ideą tego rozdziału jest pokazanie, że pisanie programów jest sportem zespołowym. A żeby odnieść sukces w zespole inżynierskim — czy też w dowolnej innej formie twórczej współpracy — musisz przeorganizować swoje zachowania pod kątem trzech głównych zasad: pokory, szacunku i zaufania.

Nie wyprzedzajmy jednak faktów i zacznijmy od przyjrzenia się temu, jak na ogół zachowują się programiści.

Kup książkę Poleć książkę

POMÓŻ MI UKRYĆ MÓJ KOD

Przez ostatnie dziesięć lat obaj wielokrotnie wygłaszaliśmy prelekcje na konferencjach programistycznych. Po uruchomieniu w 2006 roku w Google usługi open source o nazwie Project Hosting otrzymywaliśmy mnóstwo pytań i sugestii dotyczących gotowego produktu. W połowie 2008 roku zauważyliśmy pewien trend, który wyraźnie rysował się w rodzaju otrzymywanych zapytań:

> *Czy moglibyście wprowadzić w Subversion w ramach Google Code możliwość ukrywania wybranych gałęzi?*
>
> *Czy moglibyście dać możliwość tworzenia projektów open source, które początkowo są całkowicie niewidoczne na zewnątrz i stają się dostępne dopiero, gdy są gotowe?*
>
> *Cześć, chciałem przepisać kod od zera. Czy moglibyście wymazać całą historię projektu?*

Dostrzegasz wspólny motyw tych próśb?

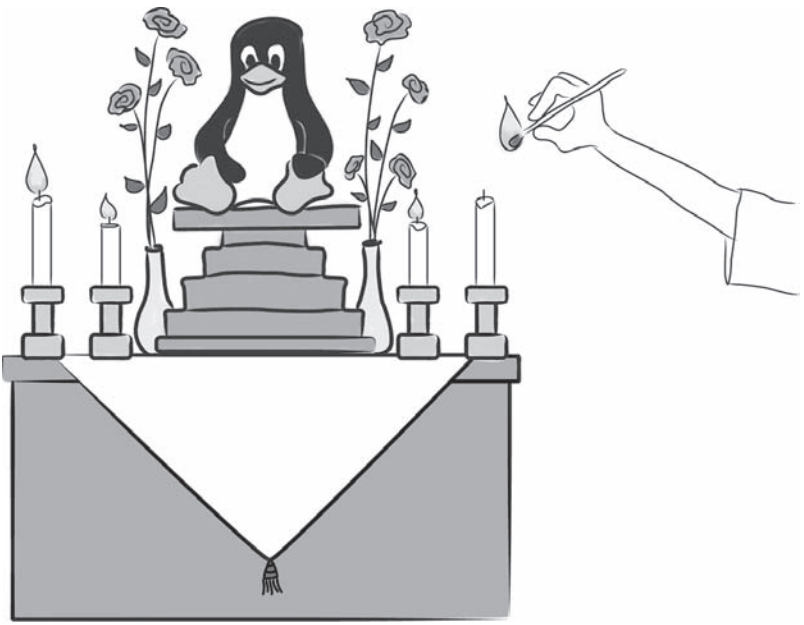
Odpowiedź brzmi *niepewność*. Ludzie boją się, że inni zobaczą ich nieukończoną pracę i zaczną ją oceniać. W jakimś sensie jest to normalny aspekt ludzkiej natury — nikt nie lubi być krytykowany, zwłaszcza za rzeczy, które nie zostały jeszcze skończone. To nastawienie rzuciło nam jednak trochę światła na pewien bardziej ogólny trend w tworzeniu oprogramowania. Owa niepewność jest bowiem symptomem większego problemu.

MIT GENIUSZA

W latach 90. obaj mieszkaliśmy w Chicago i chcąc nie chcąc byliśmy świadkami fenomenalnego pasma zwycięstw zespołu Chicago Bulls. Krajowe media przez lata pełne były opowieści na temat tej fantastycznej drużyny koszykarskiej. Tylko na czym tak naprawdę skupiały się gazety i telewizja? Nie tyle na zespole, co na Michaelu Jordanie, supergwieździe. Każdy koszykarz na świecie chciał *być* jak MJ. Patrzyliśmy jak tanecznym krokiem ogrywa innych graczy. Oglądaliśmy go w reklamach telewizyjnych. Chodziliśmy do kina na zabawne filmy, w których grał w koszykówkę z animowanymi postaciami z kreskówek. Był gwiazdą i każdy dzieciak ćwiczący wsady do kosza na miejscowym boisku skrycie marzył, że kiedyś pójdzie w jego ślady.

Kup książkę Poleć książkę

Programiści kierują się tym samym instynktem — szukają sobie idoli i czczą ich. Linus Torvalds, Richard Stallman, Bill Gates — wszyscy ci bohaterowie, którzy odmienili świat dzięki heroicznym czynom. Bo przecież Linus sam napisał Linuxa, prawda?



Tak naprawdę Linus napisał tylko wstęp, tak zwany *proof of concept* funkcjonalnego jądra uniksopodobnego, i zamieścił go na liście dyskusyjnej. Owszem, nie było to proste zadanie — to bezsprzecznie znaczące osiągnięcie — ale stanowiło ono jedynie wierzchołek góry lodowej. Linux stał się setki razy większy od tego wstępnego zarysu i jest rozwijany przez bardzo wielu błyskotliwych ludzi. Prawdziwym sukcesem Linusa było *poprowadzenie* tego zespołu i skoordynowanie jego działań; Linux jest zdumiewającym rezultatem tej współpracy. (Sam Unix zaś został napisany przez grupę równie błyskotliwych ludzi w firmie Bell Labs, a nie tylko przez Kena Thompsona i Dennisa Ritchiego).

Na tej samej zasadzie można zapytać, czy Stallman osobiście napisał wszystkie programy w ramach Free Software Foundation. Stworzył on pierwszą generację Emacsa. Ale setki innych ludzi brało udział w pisaniu basha, narzędzi GCC i całej reszty oprogramowania działającego na Linuxie.

Kup książkę Poleć książkę

Steve Jobs prowadził zespół odpowiedzialny za powstanie Macintosha, a choć Bill Gates słynie z tego, że był autorem interpretera języka BASIC dla pierwszych komputerów domowych, znacznie większym jego osiągnięciem było stworzenie prężnej firmy wokół projektu MS-DOS. Wszyscy oni stali się jednak liderami i symbolami pewnych wspólnych osiągnięć.

A co z Michaelem Jordanem?

To ta sama historia. Idealizujemy go, ale przecież tak naprawdę nie wygrał wszystkich meczów sam. Jego prawdziwy geniusz polegał na umiejętności *współpracy* z zespołem. Trener drużyny, Phil Jackson, był niezwykle mądry — jego metody szkoleniowe są wręcz legendarne. Wiedział, że jeden gracz nigdy nie zdobywa mistrzostwa sam, zmontował więc wokół MJ cały dream team. Zespół działał jak dobrze naoliwiona maszyna — był co najmniej równie godny uznania jak sam Michael.

Dlaczego więc we wszystkich tych przypadkach robimy idoli z jednostek? Dlaczego ludzie wybierają produkty polecane przez celebrytów? Czemu chcemy kupować sukienki sygnowane przez Michelle Obamę czy buty Michaela Jordana?

Ogromną rolę odgrywa w tym popularność. Ludzie mają naturalną skłonność do wyszukiwania liderów i wzorów postępowania, do czczenia ich i podejmowania prób ich naśladowania. Wszyscy potrzebujemy bohaterów, którzy by nas inspirowali — świat programowania także ma swoich herosów. Zjawisko technocelebryty nabrało cech niemal mitycznych. Wszyscy chcemy napisać coś, co zmieni świat (jak Linux), albo opracować kolejny, fenomenalny język programowania.

W głębi duszy wszyscy chcemy być geniuszami. Największym marzeniem każdego geeka jest wpadnięcie na pomysł jakiegoś przełomowego rozwiązania, by potem zaszyć się na całe tygodnie albo miesiące niczym Batman w swojej jaskini i dopieszczać do perfekcji praktyczną implementację rewolucyjnej idei. Potem zaś wystarczy pokazać nowy program światu, by zaszokować wszystkich swoim geniuszem. Koledzy są pod wrażeniem Twojej błyskotliwości. Ludzie ustawiają się w kolejki po Twój program. A za tym naturalnie idą pieniądze i sława.

Ale chwileczkę… wróćmy na ziemię. Zapewne nie jesteś geniuszem.

Nie obraź się, oczywiście — jesteśmy pewni, że jesteś bardzo inteligentnym facetem albo inteligentną dziewczyną. Ale czy zdajesz sobie sprawę, jak rzadko rodzą się *prawdziwi* geniusze? Pewnie, piszesz kod, a to niełatwa umiejętność, która sama w sobie plasuje Cię pod względem intelektualnym gdzieś powyżej średniej dla ludzkiej populacji. Nawet jeśli jednak

Kup książkę Poleć książkę

rzeczywiście jesteś geniuszem, to okazuje się, że *to nie wystarczy*. Geniusze też popełniają błędy, a nawet najbardziej błyskotliwy pomysł i najwybitniejsze umiejętności programistyczne nie gwarantują, że Twój program stanie się przebojem. Tym, od czego zależy przyszłość Twojej kariery, jest umiejętność współpracy.

Okazuje się, że mit geniusza jest po prostu kolejnym aspektem braku pewności siebie. Większość programistów boi się pokazywać ledwie rozpoczętą pracę, bo oznacza to, że koledzy dostrzegą ich błędy i będą wiedzieli, że autor kodu *nie jest geniuszem*. Przytoczmy słowa programisty z bloga Bena:

> *Zdaję sobie sprawę, że mam ogromne opory przed pokazywaniem ludziom nieukończonego projektu. Wydaje mi się, że będą mnie krytykować i uznają mnie za idiotę.*

To niezwykle powszechna wśród programistów postawa, a naturalną reakcją na nią jest chęć zaszycia się w jaskini i praca, praca, praca. Nikt wtedy nie zobaczy Twoich potknięć i będziesz mieć szansę zaprezentować swoje dzieło, gdy będzie już skończone. Trzeba je schować przed światem, dopieścić do perfekcji.

Kolejnym częstym powodem dążenia do ukrywania swoich kart jest lęk przed tym, że jakiś inny programista przejmie Twój pomysł i zacznie go realizować, zanim Ty zdążysz się za to zabrać. Trzymając pomysł w sekrecie, masz nad nim pełną kontrolę.

Wiemy, co sobie zapewne teraz myślisz: no i co z tego? Czy ludzie nie powinni pracować tak, jak chcą?

Otóż nie. Zapewniamy Cię, że w tym konkretnym przypadku postępujesz źle i to *naprawdę* poważna sprawa. Już tłumaczymy dlaczego.

UKRYWANIE JEST UZNAWANE ZA SZKODLIWE

Jeśli przez cały czas pracujesz sam, *zwiększasz* ryzyko błędu i tamujesz potencjał własnego rozwoju.

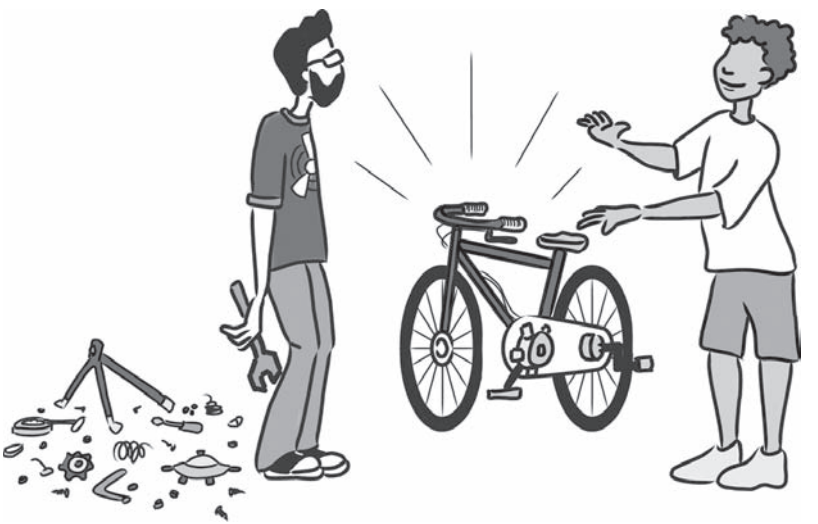
Zacznijmy od tego, skąd wiesz, że w ogóle jesteś na właściwym tropie.

Wyobraź sobie, że jesteś zapalonym projektantem rowerów i któregoś dnia wpadasz na genialny pomysł nowej, przełomowej konstrukcji przerzutki. Zamawiasz części i przez całe tygodnie tkwisz w warsztacie, próbując zbudować prototyp. Kiedy Twój sąsiad — też miłośnik dwóch kółek — pyta, co słychać, postanawiasz nie zdradzać się ze swoim pomysłem.

Kup książkę Poleć książkę

Nie chcesz, aby ktokolwiek dowiedział się o przedsięwzięciu, zanim nie stworzysz produktu doskonałego. Mijają miesiące, a Ty masz problemy z doprowadzeniem prototypu do stanu używalności. Pracujesz jednak w tajemnicy, nie możesz więc liczyć na porady uzdolnionych technicznie przyjaciół.

Któregoś dnia Twój sąsiad wystawia z garażu rower z zupełnie przeprojektowanym mechanizmem przerzutek. Okazuje się, że konstruował coś bardzo podobnego do Twojego wynalazku, ale z pomocą przyjaciół z miejscowego sklepu rowerowego. Wpadasz w rozpacz. Prezentujesz mu swoją pracę. Sąsiad dostrzega kilka prostych błędów w projekcie — takich, które można byłoby usunąć w pierwszym tygodniu pracy, gdybyś oczywiście pokazał mu, czym się zajmujesz.



Z tej opowieści płynie kilka lekcji. Jeśli ukrywasz jakiś znakomity pomysł przed światem i nie chcesz go pokazywać nikomu, dopóki nie doszlifujesz całości, to podejmujesz ogromne ryzyko. Na początku bardzo łatwo jest popełnić fundamentalne błędy. Ryzykujesz, że wynajdziesz koło na nowo[1]. Poza tym nie czerpiesz korzyści ze współpracy: zauważyłeś, o ile szybciej zrealizował swój zamiar sąsiad dzięki pracy z innymi? To dlatego ludzie maczają palec w wodzie, zanim do niej wskoczą: trzeba mieć pewność,

[1] I to dosłownie, jeśli rzeczywiście projektujesz rowery.

Kup książkę Poleć książkę

że pracuje się nad właściwą rzeczą, że robi się to prawidłowo i że ktoś nie zrobił tego już wcześniej. Potknięcie przy stawianiu pierwszych kroków jest bardzo prawdopodobne. Im więcej opinii zwrotnych zgromadzisz na początku, tym mniejsze staje się to ryzyko[2]. Zapamiętaj sprawdzoną zasadę: myl się wcześnie, myl się szybko, myl się często — o roli pomyłek i potknięć będziemy obszernie pisać w dalszej części tej książki.

We wczesnym dzieleniu się pomysłami nie chodzi tylko o to, by zweryfikować ich słuszność i uniknąć potknięć. Ważne jest także zwiększenie wartości czegoś, co nazywamy *współczynnikiem autobusu* dla Twojego projektu.

Współczynnik autobusu — liczba osób, które musiałby przejechać autobus, żeby Twoje przedsięwzięcie zostało skazane na porażkę.

Jak rozproszona jest wiedza i know-how w Twoim przedsięwzięciu? Jeżeli jesteś jedyną osobą, która rozumie mechanikę działania kodu prototypu, to stanowi to przyjemną gwarancję bezpieczeństwa, jeśli chodzi o utrzymanie się w pracy, ale oznacza to zarazem, że jeżeli przejedzie Cię autobus, projekt umrze wraz z Tobą. Jeśli jednak nawiążesz współpracę

[2] Należy podkreślić, że czasami zbyt duża liczba opinii zwrotnych na pierwszym etapie procesu twórczego jest ryzykowna, ale tym zajmiemy się w jednym z kolejnych rozdziałów.

Kup książkę Poleć książkę

z przyjacielem, współczynnik autobusu podwoi się. A jeśli masz mały zespół, który wspólnie projektuje i pracuje nad prototypem, sprawa ma się jeszcze lepiej — projekt nie przepadnie, nawet jeśli któryś z członków zespołu zniknie. Pamiętaj, że członkowie zespołu wcale nie muszą dosłownie wpadać pod autobus — mogą się im przydarzyć inne, nieprzewidziane życiowe przypadki. Ktoś może wziąć ślub, ktoś zostanie zmuszony do przeprowadzki, ktoś inny zwolni się z pracy albo będzie musiał się zająć bliskim, który zachorował. Musisz zadbać o współczynnik autobusu, aby zabezpieczyć przyszły sukces przedsięwzięcia.

Oprócz współczynnika autobusu w grę wchodzi kwestia tempa realizacji przedsięwzięcia. Łatwo zapomnieć, że samotna praca często jest mozolną harówką, znacznie wolniejszą, niż przyznałaby większość ludzi. Jak często się uczysz podczas samodzielnej pracy? Jak szybko działasz? Internet jest fantastycznym polem wymiany opinii i informacji, ale nie zastąpi rzeczywistego kontaktu z ludźmi. Bezpośrednia współpraca przyczynia się do wzrostu kolektywnej wiedzy stojącej za Waszymi wysiłkami. Jeśli utkwisz na jakiejś absurdalnej przeszkodzie, ile zmarnujesz czasu, zanim uda Ci się ją pokonać? Pomyśl, o ile inaczej by to wyglądało, gdybyś miał kilku kolegów, którzy zerkną Ci przez ramię i powiedzą — od ręki — że dałeś plamę, i wyjaśnią, jak rozwiązać problem. Właśnie dlatego w firmach zajmujących się produkcją oprogramowania zespoły pracują wspólnie (albo programują dwójkami). Często przydaje się bowiem druga para oczu.

A oto inna analogia. Pomyśl o tym, w jaki sposób posługujesz się kompilatorem. Kiedy przystępujesz do pisania dużego programu, czy robisz tak, że najpierw przez wiele dni piszesz dziesięć tysięcy linii kodu i dopiero potem, gdy uznasz, że zrobiłeś już wszystko i całość powinna działać absolutnie bez zarzutu, po raz pierwszy wydajesz polecenie „kompiluj”? Oczywiście, że nie. Wyobrażasz sobie, jaką by się to skończyło katastrofą? Programiści pracują najsprawniej w cyklach zapewniających *błyskawiczną* informację zwrotną. Piszemy nową funkcję i kompilujemy. Dodajemy test funkcjonalny i kompilujemy. Refaktoryzujemy fragment kodu i kompilujemy. Literówki i błędy korygujemy najszybciej jak to możliwe po wygenerowaniu kodu. Chcemy, aby kompilator towarzyszył nam przy każdym, najmniejszym kroku i grał rolę skrzydłowego; w niektórych środowiskach programistycznych kod może być kompilowany nawet *w trakcie pisania*. Dzięki temu możemy liczyć na wysoką jakość kodu i mieć pewność, że nasz program rozwija się prawidłowo, krok po kroku.

Kup książkę Poleć książkę

Analogiczny rodzaj cyklu zapewniającego szybką informację zwrotną jest potrzebny nie tylko na poziomie kodu, ale też na poziomie całego projektu. Ambitne przedsięwzięcia szybko ewoluują i muszą się adaptować do zmieniającego się środowiska. Wpadają na nieprzewidywane przeszkody projektowe, stają się przedmiotem politycznych zatargów albo po prostu okazuje się, że pewne sprawy nie działają w nich tak jak powinny. Wymagania nieoczekiwanie się zmieniają. Jak zapewnić sobie taki cykl informacji zwrotnych, aby jak najszybciej zdać sobie sprawę z konieczności zmiany planów? Odpowiedź brzmi: dzięki pracy zespołowej. Często cytuje się Erica Raymonda, który miał powiedzieć: „Wiele oczu wyłapie każdy błąd" (ang. *many eyes make all bugs shallow*), ale może trafniej byłoby to ująć następująco: „Wiele oczu zadba o użyteczność i właściwy kierunek rozwoju projektu". Ludzie pracujący w jaskiniach często nagle budzą się i odkrywają, że choć ich pierwotna wizja była słuszna, świat się zmienił, a ich produkt stał się bezużyteczny.

O inżynierach i o biurach

Dwadzieścia lat temu powszechnie sądziło się, że jeśli inżynier ma być produktywny, to trzeba mu zapewnić własny gabinet z zamkniętymi drzwiami. Miał to być jedyny sposób, by wygospodarować duże przedziały czasu na niezakłócone przez nikogo skupienie się na pisaniu obszernych fragmentów kodu.

Naszym zdaniem większości inżynierów[3] prywatny gabinet jest nie tylko niepotrzebny, ale wręcz stanowi zagrożenie. Dziś programy piszą zespoły, nie jednostki, a łatwo dostępna możliwość szerokopasmowej komunikacji z resztą zespołu jest nawet cenniejsza niż łącze internetowe.

Możesz mieć mnóstwo niezakłóconego niczym czasu, ale jeśli spożytkujesz go na pracę nad *niewłaściwą rzeczą*, to będzie to czas zmarnowany.

Niestety wydaje się, że dzisiejsze firmy technologiczne popadły w drugą skrajność. Po wejściu do biura okazuje się, że inżynierowie

[3] Zdajemy sobie jednocześnie sprawę z tego, że niektórym zatwardziałym introwertykom potrzeba więcej spokoju, ciszy i samotności niż większości ludzi. Takim osobom może sprzyjać zaciszniejsze otoczenie, a być może nawet własny gabinet.

Kup książkę Poleć książkę

tkwią w gigantycznych pomieszczeniach — na 50 albo 100 osób — bez jakichkolwiek ścian i przegród. Kwestia takiej otwartej przestrzeni jest teraz przedmiotem gorących dyskusji. Każda, nawet najbłahsza wymiana zdań staje się w takim pomieszczeniu publiczna, a ludzie przestają ze sobą rozmawiać, by nie irytować dziesiątek współpracowników. To rozwiązanie jest równie złe jak prywatne gabinety!

Naszym zdaniem najlepszym wyjściem jest rozwiązanie pośrednie: podzielić pracowników na zespoły liczące od 6 do 12 osób, zebrane w małych pokojach (albo dużych biurach), aby można było ze sobą rozmawiać spontanicznie (i bez oporów).

Oczywiście w pewnych sytuacjach pojedynczy inżynierowie wciąż muszą mieć możliwość odcięcia się od szumu i zakłóceń — to dlatego większość zespołów, z jakimi mieliśmy do czynienia, wypracowała jakiś sposób sygnalizowania, że ktoś jest aktualnie zajęty i trzeba do minimum ograniczyć przerywanie mu. Pracowaliśmy kiedyś w zespole, w którym funkcjonowała procedura „przerw głosowych": jeśli chciało się pogadać, mówiło się „Mary, przerwa", gdzie *Mary* było imieniem osoby, z którą chciałeś porozmawiać. Jeśli Mary była na etapie, na którym mogła zrobić sobie przerwę, to odwracała się na krześle i zaczynała słuchać. A jeśli była zajęta, odpowiadała tylko „yhy" i dalej zajmowała się swoimi sprawami, aż dokończyła to, co miała w danej chwili na głowie.

W innych zespołach inżynierom udostępnia się słuchawki z układem aktywnego filtrowania szumu otoczenia, aby ułatwić im wytłumienie zewnętrznego chaosu — co więcej, w wielu firmach już sam fakt założenia słuchawek jest równoznaczny z powiedzeniem „nie przeszkadzaj mi, chyba że to naprawdę ważne". W jeszcze innych zespołach funkcjonuje system symboli albo pluszowych zwierzaków, które członkowie grupy roboczej stawiają na monitorach, aby zaznaczyć, że wolno im przeszkadzać wyłącznie w razie najwyższej konieczności.

Nie zrozum nas źle — nadal jesteśmy zdania, że inżynierowie potrzebują spokoju i czasu, aby móc się skupić na pisaniu kodu, ale sądzimy zarazem, że w równej mierze potrzebują szerokopasmowej i płynnej komunikacji z zespołem. Znalezienie złotego środka jest sztuką samą w sobie.

Kup książkę Poleć książkę

Wszystko to sprowadza się do następującego stwierdzenia: *praca samotna wiąże się z większym ryzykiem niż praca z innymi*. Możesz się wprawdzie obawiać, że ktoś ukradnie Twój pomysł albo pomyśli, że jesteś głupi, ale o wiele bardziej należy się bać zmarnowania mnóstwa czasu na zajmowanie się niewłaściwymi sprawami.

Niestety problem z chomikowaniem pomysłów nie ogranicza się do inżynierii oprogramowania — występuje powszechnie w różnych dziedzinach. Na przykład na polu nauk ścisłych *powinna* panować wolna i swobodna wymiana informacji. Tymczasem rozpaczliwe trzymanie się zasady „publikuj albo giń" i rywalizacja o granty przynoszą dokładnie odwrotny skutek. Wybitni naukowcy nie dzielą się pomysłami. Obsesyjnie się z nimi kryją, prowadzą indywidualne badania, tuszują wszystkie popełnione po drodze błędy, by wreszcie opublikować pracę, starając się stworzyć wrażenie, że przyszło im to bez trudu i było oczywiste. Tymczasem rezultaty często są katastrofalne: przypadkiem powielają czyjeś dokonania, już na początku popełniają i przegapiają jakiś błąd albo dochodzą do czegoś, co mogło być ciekawe wcześniej, ale w chwili publikacji jest już bezużyteczne. Ilość zmarnowanego czasu i wysiłku woła o pomstę do nieba.

Nie bądź kolejnym, który uprawia sztukę dla sztuki.

LICZY SIĘ TYLKO ZESPÓŁ

Cofnijmy się zatem i połączmy wszystkie przedstawione pomysły.

Koncepcja, którą z takim uporem forsujemy, polega na tym, że w świecie programowania samotni rzemieślnicy zdarzają się bardzo rzadko — a nawet jeśli istnieją, nie dochodzą do nadludzkich rezultatów w próżni; ich przełomowe dokonania niemal zawsze są skutkiem heroicznego wysiłku zespołowego będącego pokłosiem iskry inspiracji.

Prawdziwym wyzwaniem jest stworzenie wokół supergwiazdy *zespołu* — i jest to diabelnie trudne zadanie. Najlepsze zespoły robią znakomity użytek ze swoich supergwiazd, ale jako całość są zawsze wybitniejsze niż suma ich poszczególnych części.

Ujmijmy to prościej:

Tworzenie oprogramowania jest zawsze sportem zespołowym.

Początkowo trudno się z tą koncepcją pogodzić, bo stoi ona w jawnej sprzeczności z mitem genialnego programisty. Spróbuj ją sobie jednak powtarzać jak mantrę.

Kup książkę Poleć książkę

Nie wystarczy być bystrym, jeśli samotnie tkwi się w swojej hakerskiej kryjówce. Nie zmienisz świata albo nie zachwycisz milionów użytkowników komputerów, przygotowując jakiś wynalazek w sekrecie. Musisz pracować z innymi ludźmi. Dzielić się swoją wizją. Dzielić się pracą. Uczyć się od innych. Stworzyć błyskotliwy zespół.

Pomyśl tylko: ile potrafisz wymienić bardzo popularnych, użytecznych programów, które zostały od początku do końca napisane przez *jednego* człowieka? (Niektórzy powiedzą może: „LaTeX", ale trudno nazwać go „bardzo popularnym" — chyba że uznać grupę ludzi piszących prace naukowe za statystycznie istotną część wszystkich użytkowników komputerów!).

Do koncepcji sportu zespołowego będziemy wracać w tej książce raz po raz. Sprawnie funkcjonujące zespoły są głównym i najprawdziwszym kluczem do sukcesu. Powinieneś do tego dążyć na wszelkie możliwe sposoby — właśnie temu jest poświęcona cała ta książka.

TRZY FILARY

Uzasadniliśmy sens pracy zespołowej. Jeśli zatem praca zespołowa jest najlepszym sposobem na tworzenie wybitnego oprogramowania, to jak stworzyć (albo znaleźć) doskonały zespół?

To nie takie proste. Aby osiągnąć kolektywną nirwanę, należy najpierw poznać i przyswoić coś, co nazywamy trzema zasadami, czy też filarami

Kup książkę Poleć książkę

umiejętności interpersonalnych. Te trzy zasady nie sprowadzają się tylko do ułatwiania wzajemnych kontaktów; stanowią one fundament, na którym bazują wszelkie zdrowe relacje i formy współpracy.

Pokora

Nie jesteś pępkiem świata. Nie jesteś wszechwiedzący ani nieomylny. Jesteś otwarty na możliwość samorozwoju.

Szacunek

Szczerze troszczysz się o kolegów, z którymi pracujesz. Traktujesz ich jak ludzi, doceniasz ich umiejętności i osiągnięcia.

Zaufanie

Wierzysz, że inni ludzie są kompetentni i zrobią to, co do nich należy. W uzasadnionych sytuacjach pozwalasz im kierować[4].

Łącznie zasady te nazywamy akronimem HRT, od angielskich nazw owych filarów: *humility*, *respect*, *trust*. Akronim ten wymawiamy jak słowo *heart* (ang. serce), a nie *hurt* (ang. krzywdzić), bo chodzi o to, by *ograniczyć* bolesne problemy, a nie ranić innych. Co więcej, na tych filarach jest zbudowana nasza główna teza:

Niemal każdy konflikt społeczny ma swoje pierwotne źródło w braku pokory, szacunku albo zaufania.

Początkowo może Ci się to wydawać mało prawdopodobne, ale daj tej teorii szansę. Pomyśl o jakiejś nieprzyjemnej albo niezręcznej sytuacji międzyludzkiej, w której się teraz znajdujesz. Czy na absolutnie fundamentalnym poziomie wszyscy przejawiają dostateczną pokorę? Czy ludzie naprawdę szanują siebie nawzajem? Czy mają do siebie zaufanie?

Do tego stopnia wierzymy w istotność tych filarów, że oparliśmy na nich nawet strukturę tej książki.

Ta książka zaczyna się od Ciebie — od przyjęcia przez Ciebie zasad HRT i prawdziwie głębokiego zrozumienia tego, co oznacza umieszczenie HRT w centrum swoich relacji. Pierwszy rozdział jest poświęcony właśnie temu. I stąd rozchodzą się zataczające coraz szersze kręgi fale wpływów owych zasad.

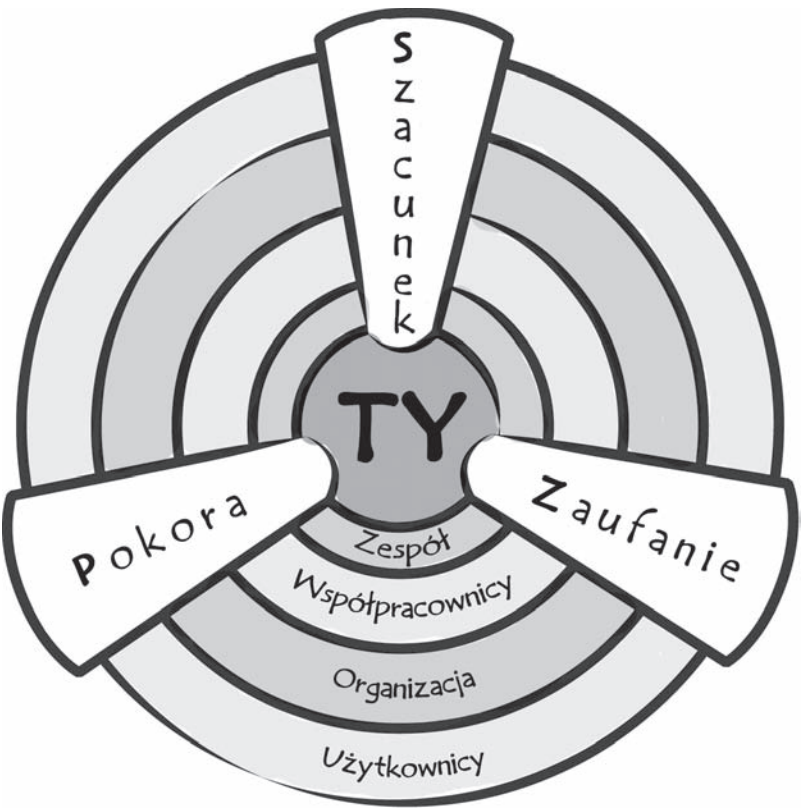
---

[4] Co jest niezwykle trudne, jeśli kiedyś sparzyłeś się na delegowaniu zadań niekompetentnym ludziom.

Kup książkę Poleć książkę

W rozdziale 2. omawiamy wyzwania związane z budowaniem zespołu na bazie tych trzech filarów. Stworzenie kultury zespołu jest niezwykle ważnym kolejnym krokiem do sukcesu — chodzi o dream team, o którym była mowa wcześniej.

Następnie przyjrzymy się ludziom, którzy na co dzień kontaktują się z Twoim zespołem, ale nie muszą wchodzić w ścisły krąg jego kultury. Mogą to być współpracownicy z innych zespołów albo wolontariusze oferujący pomoc w pracy nad przedsięwzięciem. Wielu z nich nie tylko nie przestrzega zasad HRT, ale może być wręcz toksycznych! Priorytetową sprawą jest opanowanie umiejętności bronienia zespołu przed takimi ludźmi. Prawdziwym i ostatecznym celem powinno być jednak usunięcie im kłów jadowych i wciągnięcie do kultury Twojego zespołu. To znakomity sposób na jego rozszerzenie.



Kup książkę Poleć książkę

Większość zespołów funkcjonuje w ramach dużej firmy, której środowisko często bywa takim samym utrudnieniem jak toksyczni ludzie. Opanowanie sztuki omijania organizacyjnych raf może decydować o tym, czy produkt ujrzy światło dzienne, czy zostanie skreślony.

Na koniec weźmiemy pod uwagę użytkowników Twoich programów. Zdarza się nam zapomnieć, że istnieją, choć w istocie stanowią życiodajną siłę każdego przedsięwzięcia. Bez użytkowników istnienie Twojego programu jest pozbawione sensu. Te same zasady HRT, które kwitną w Twoim zespole, musisz wykorzystywać w kontaktach z użytkownikami — takie postępowanie przynosi ogromne korzyści.

Zróbmy krótką przerwę.

Kiedy sięgałeś po tę książkę, zapewne nie sądziłeś, że decydujesz się na coś w rodzaju udziału w cotygodniowych spotkaniach grupy wsparcia. Rozumiemy Cię. Zmaganie się z problemami społecznymi bywa trudne. Ludzie są chaotyczni, nieprzewidywalni i często irytujący w kontaktach. Każdego z nas kusi, by zamiast wkładać energię w analizowanie układów międzyludzkich i podejmowanie strategicznych działań machnąć na to wszystko ręką. Znacznie łatwiej się dogadać z przewidywalnym kompilatorem, prawda? Po co więc zawracać sobie głowę aspektami interpersonalnymi?

Oto cytat ze znanej publikacji autorstwa Richarda Hamminga[5]:

> *Dzięki wysiłkowi włożonemu w opowiadanie dowcipów sekretarkom i odrobinie sympatii mogłem liczyć na wyjątkową pomoc z ich strony. Na przykład któregoś razu, z jakiegoś idiotycznego powodu, nie działała w Murray Hill ani jedna kserokopiarka. Nie pytaj mnie dlaczego. Miałem coś do zrobienia. Moja sekretarka zadzwoniła do kogoś w Holmdel[6], wskoczyła do firmowego auta, wybrała się w godzinną podróż, skserowała co trzeba i wróciła. To był rewanż za czasy, kiedy starałem się ją rozweselać, opowiadałem jej dowcipy i zachowywałem się przyjaźnie; ta odrobina fatygi sowicie mi się potem opłaciła. Dzięki uświadomieniu sobie, że musisz pracować w ramach pewnego systemu, i dzięki zgłębianiu tego, jak może ci on pomóc w wykonaniu pracy, uczysz się adaptować ów system do swoich oczekiwań.*

---

[5] R. Hamming, „You and Your Research", *http://bit.ly/hamming_paper*.

[6] W Murray Hill i Holmdel mieściły się oddziały firmy Bell Labs, w której pracował Richard Hamming, wybitny amerykański matematyk i pionier nauk komputerowych — *przyp. tłum.*

Kup książkę Poleć książkę

Morał jest następujący: warto docenić możliwości, jakie daje zaangażowanie się w społeczną grę. Nie chodzi o to, aby oszukiwać ludzi albo by nimi manipulować, lecz o tworzenie relacji pozwalających wykonywać zadania, a relacje *zawsze* są trwalsze od pojedynczych przedsięwzięć.

HRT W PRAKTYCE

Całe to rozprawianie o pokorze, szacunku i zaufaniu brzmi jak materiał na kazanie. Zejdźmy zatem na ziemię i zastanówmy się, jak zastosować te koncepcje w realnych sytuacjach. Szukamy praktycznych sugestii, przeanalizujemy więc listę konkretnych zachowań i przykładów, od których możesz zacząć. Wiele z nich może początkowo wydawać się oczywistych, ale odkąd zaczniesz zwracać na nie uwagę, przekonasz się, jak często Ty i Twoi koledzy *nie* postępujecie w myśl owych oczywistości.

Utemperuj ego

To nieco inny, łatwiejszy sposób na powiedzenie komuś, kto nie ma w sobie dostatecznie wiele *pokory*, aby trochę spuścił z tonu. Nikt nie chce pracować z kimś, kto nieustannie zachowuje się, jakby był pępkiem świata. Nawet jeśli masz świadomość, że jesteś najmądrzejszym człowiekiem biorącym udział w dyskusji, nie okazuj tego wprost. Czy na przykład zawsze musisz mieć ostatnie słowo na każdy temat? Albo pierwsze? Czy czujesz potrzebę komentowania każdego szczegółu jakiejś propozycji albo debaty? A może znasz kogoś, kto tak postępuje?

Zauważ, że bycie pokornym *nie* oznacza, iż należy się przed wszystkimi płaszczyć; w pewności siebie nie ma nic złego. Po prostu nie zgrywaj wszechwiedzącego. A jeszcze lepiej: pomyśl o połechtaniu „kolektywnego ego" — zamiast zastanawiać się nad tym, jaki jesteś wspaniały, spróbuj zbudować poczucie zespołowego sukcesu i dumy. Na przykład Apache Software Foundation może się poszczycić długą historią tworzenia wspólnot wokół projektów programistycznych; wspólnoty te mają bardzo silną tożsamość i odrzucają osoby, którym bardziej zależy na promowaniu samych siebie.

Ego przejawia się na wiele sposobów, które bardzo często szkodzą produktywności i spowalniają. Oto kolejna zaczerpnięta z publikacji Hamminga świetna opowieść, która doskonale ilustruje to stwierdzenie:

Kup książkę Poleć książkę

John Tukey niemal zawsze ubierał się bardzo swobodnie. Wchodził do gabinetu jakiejś szychy i długą chwilę zajmowało, nim jego rozmówca orientował się, że ma do czynienia ze znakomitością i lepiej, żeby nadstawił uszu. Przez długi czas John musiał borykać się z tego rodzaju wrogością. Ileż to zmarnowanego wysiłku! Nie twierdzę, że powinieneś się dostosowywać; powiedziałem tylko, że „gotowość do dostosowania przynosi wielkie korzyści". Jeśli na różne sposoby dążysz do afiszowania się ze swoim ego („zamierzam zrobić to po swojemu"), będziesz ponosić niewielkie, stałe koszty przez cały czas trwania kariery zawodowej. To zaś, z perspektywy całego życia, kumuluje się do wielkiej sterty niepotrzebnych kłopotów. […] Dzięki uświadomieniu sobie, że musisz pracować w ramach pewnego systemu, i dzięki zgłębianiu tego, jak może ci on pomóc w wykonaniu pracy, uczysz się adaptować ów system do swoich oczekiwań. Możesz też nieustannie z nim walczyć, jakbyś przez całe życie toczył małą, niewypowiedzianą wojnę.

Ucz się krytykować i być krytykowanym

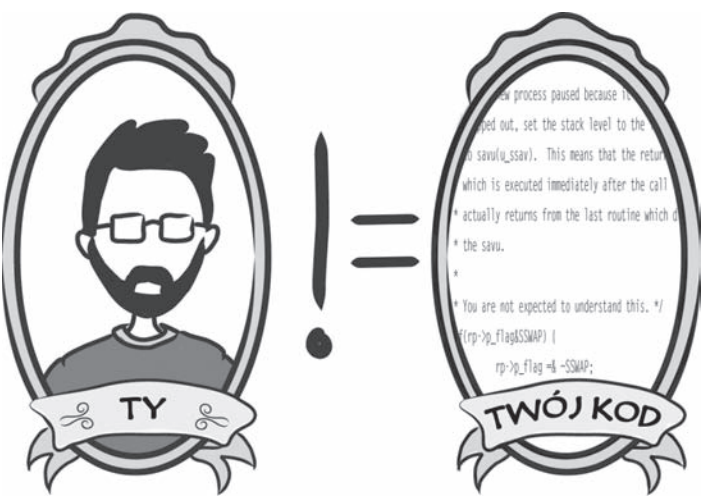
Programista Joe przyszedł do nowej pracy. Po pierwszym tygodniu już całkiem sprawnie orientował się w bazie kodu. Ponieważ zależało mu na firmie, zaczął łagodnie podpytywać współpracowników o ich wkład w rozwój projektu. Wysyłał e-mailami krótkie oceny fragmentów kodu, kulturalnie pytał o założenia projektowe albo wskazywał miejsca, w których dałoby się usprawnić logikę programu. Po kilku tygodniach został wezwany do gabinetu dyrektora. „W czym problem?" — spytał Joe. „Czy zrobiłem coś nie tak?". Dyrektor przyjrzał się mu z zaniepokojeniem. „Mieliśmy mnóstwo zarzutów odnośnie do twojego zachowania, Joe. Wygląda na to, że byłeś bardzo niemiły wobec współpracowników, bo krytykowałeś ich na lewo i prawo. Są źli. Powinieneś trochę spuścić z tonu". Joe był w kompletnym szoku. W kulturze mocno osadzonej na filarach HRT opinie Joego dotyczące kodu byłyby oczekiwane i mile widziane. W tym przypadku jednak Joe powinien był wykazać się nieco większą wrażliwością na szerzącą się w zespole niepewność i znaleźć subtelniejszy sposób na wdrożenie w kulturze firmowej zwyczaju inspekcji kodu.

W świecie profesjonalnego tworzenia oprogramowania krytyka niemal nigdy nie ma osobistego charakteru — zwykle jest to po prostu element procesu mającego na celu stworzenie lepszego produktu. Sztuka polega na zrozumieniu (przy czym dotyczy to w równej mierze Ciebie i otaczających

Kup książkę Poleć książkę

Cię ludzi) różnicy między konstruktywną krytyką czyichś twórczych działań a wycieczkami osobistymi mającymi na celowniku czyjś charakter. To drugie jest bezużyteczne, uciążliwe i praktycznie nie da się przed tym bronić. To pierwsze jest zaś zawsze pomocne i daje wskazówki umożliwiające doskonalenie się. Co najważniejsze zaś, jest nacechowane *szacunkiem*: osoba udzielająca konstruktywnej krytyki szczerze troszczy się o drugiego człowieka i zależy jej na doskonaleniu się kolegi albo jego pracy. Naucz się szanować kolegów i krytykować konstruktywnie, z uprzejmością. Jeśli naprawdę kogoś szanujesz, będziesz chętniej dobierać taktowne, pomocne słowa — ta umiejętność wymaga jednak wielu ćwiczeń.

Z drugiej strony, musisz też nauczyć się przyjmować krytykę. To oznacza nie tylko *skromność* w odniesieniu do własnych umiejętności, ale też *zaufanie*, że drugi człowiek ma najlepsze intencje wobec Ciebie (i Twojego przedsięwzięcia!) i wcale nie uważa Cię za idiotę. Programowanie, jak wszystko inne, jest pewną umiejętnością. Ćwiczenie czyni mistrza. Jeśli kolega podpowiedziałby Ci jakieś triki, dzięki którym mógłbyś wznieść na wyższy poziom umiejętność żonglowania, to czy potraktowałbyś to jako atak na Twoją osobowość i wartość jako istoty ludzkiej? Mamy nadzieję, że nie. Na tej samej zasadzie *Twoje poczucie własnej wartości nie powinno być uzależnione od kodu, który piszesz — albo od dowolnego twórczego przedsięwzięcia, którego się podjąłeś.* Powtórzymy się: nie jesteś swoim kodem. Ty także to sobie powtarzaj. *Nie jesteś tym, co robisz*. Musisz nie tylko wierzyć w siebie, ale też sprawić, by Twoi współpracownicy w Ciebie wierzyli.



Kup książkę Poleć książkę

Na przykład jeśli masz współpracownika, któremu brakuje pewności siebie, to *nie* zwracaj się do niego następująco: „Chłopie, kontrola przepływu w tej metodzie jest kompletnie do niczego. Tak jak wszyscy inni, powinieneś zastosować standardowy wzorzec xyzzy". W tej opinii aż roi się od lapsusów: mówisz komuś, że zrobił coś „do niczego" (zupełnie jakby świat był czarno-biały!), żądasz, żeby coś zmienił, i oskarżasz go o stworzenie czegoś wbrew ogólnie przyjętym normom (przez co ów ktoś czuje się jak głupek). Reakcja będzie przesadnie emocjonalna, typowa dla człowieka przypartego do muru.

To samo można powiedzieć w lepszy sposób, na przykład: „Wiesz co, nie mogę się połapać w kontroli przepływu w tej sekcji. Zastanawiam się, czy zastosowanie wzorca xyzzy nie uprościłoby tego kodu i nie ułatwiło jego utrzymania". Zauważ, że przyjmując pokorną postawę, zmieniasz charakter kwestii, która nie dotyczy już tego współpracownika, lecz Ciebie. To nie on się myli, to Ty masz problemy ze zrozumieniem kodu. Twoja sugestia brzmi jak poszukiwanie sposobu, by uczynić problem bardziej przystępnym dla Twojej biednej, skołatanej głowy, a być może przy okazji pomogłaby w długofalowym utrzymaniu projektu. Poza tym niczego nie żądasz — dajesz współpracownikowi możliwość pokojowego odrzucenia sugestii. Dyskusja toczy się wyłącznie na płaszczyźnie kodu i nie dotyczy czyjejś wartości albo umiejętności programistycznych.

Myl się szybko i iteruj

W świecie biznesu krąży dobrze znana (i wyświechtana) miejska legenda o menedżerze, który popełnił błąd kosztujący firmę zawrotne 10 milionów dolarów. Następnego dnia pełen najczarniejszych uczuć menedżer idzie do pracy i zaczyna pakować z biurka swoje rzeczy, a gdy otrzymuje nieunikniony telefon („dyrektor generalny chce cię widzieć w swoim gabinecie"), potulnie drepcze do gabinetu szefa i smętnie podsuwa mu na biurku niewielki dokument.

— Co to takiego? — pyta dyrektor.

— Moje wypowiedzenie — odpowiada menedżer. — Rozumiem, że wezwał mnie pan, żeby mnie zwolnić.

— *Zwolnić*?! — mówi z niedowierzaniem dyrektor. — Czemu miałbym cię zwalniać? Właśnie wydałem dziesięć milionów dolarów na *twoje szkolenie*![7]

[7] W internecie krąży kilkanaście wersji tej legendy, przypisywanych różnym słynnym menedżerom.

Kup książkę Poleć książkę

Oczywiście jest to ekstremalny przypadek, ale ów dyrektor generalny rozumie, że zwolnienie menedżera nie tylko nie zrekompensuje mu 10-milionowej straty, ale na dodatek będzie miało bolesny skutek w postaci utraty cennego pracownika wysokiego szczebla, który — można być tego pewnym — zrobi wszystko, by więcej nie popełnić tego rodzaju błędu.

Jedno z ulubionych haseł w Google brzmi: pomyłka wchodzi w grę. Powszechnie uważa się bowiem, że jeśli raz na jakiś czas się nie pomylisz, to nie jesteś wystarczająco innowacyjny albo nie podejmujesz dostatecznego ryzyka. Pomyłka jest uznawana jako doskonała okazja do nauki i do poprawienia się następnym razem. Często przytacza się słowa Thomasa Edisona, który ponoć powiedział: „Jeśli znajdę dziesięć tysięcy nieskutecznych sposobów zrobienia czegoś, to nie poniosłem porażki. Nie zniechęcam się, bo każda niepomyślna próba jest kolejnym krokiem naprzód".

W Google X — dziale firmy zajmującym się futurystycznymi pomysłami, takimi jak Google Glass czy autonomiczne samochody — pomyłka jest rozmyślnie wpisana w system motywacyjny. Ludzie wpadają na zwariowane pomysły, a ich współpracowników zachęca się do jak najszybszego „odstrzeliwania" tych pomysłów i nagradza (prowadzone są nawet rywalizacje!) za obalenie jak największej liczby teorii w ustalonym czasie. Dopiero gdy jakiegoś pomysłu naprawdę nie da się przekreślić na białej tablicy pomimo starań wszystkich współpracowników, taki pomysł *ma szansę* na przejście do stadium wczesnego prototypu.

Kluczem do uczenia się na błędach jest dokumentowanie porażek. Taką dokumentację w naszej branży często określa się mianem *post mortem*. Dołóż wszelkich starań, by takie „pośmiertne" akta projektu nie były tylko listą przeprosin albo wymówek — nie taki jest ich cel. Dobre post mortem powinno zawsze zawierać opis tego, *czego się nauczyłeś i co się zmieni* dzięki tej nauczce. Gotowy dokument koniecznie umieść w łatwo dostępnym miejscu i naprawdę wykorzystaj proponowane w nim zmiany. Pamiętaj, że prawidłowa dokumentacja błędów ułatwia innym ludziom (teraz i w przyszłości) zrozumienie, co się stało, i uniknięcie powtórzenia tych samych pomyłek. Nie zacieraj śladów — oświetlaj je niczym pas startowy dla tych, którzy chcieliby iść Twoją drogą!

Dobra dokumentacja post mortem powinna obejmować:

- krótkie podsumowanie;
- historię zdarzenia, od jego zajścia, przez analizę, aż do rozwiązania;
- główną przyczynę zdarzenia;

Kup książkę Poleć książkę

- oszacowanie wpływu i szkód;
- zbiór działań mających na celu bieżące rozwiązanie problemu;
- zbiór działań mających na celu zapobiegnięcie powtórzeniu się zdarzenia w przyszłości;
- wyciągnięte wnioski.

Zostaw sobie czas na naukę

Cindy była supergwiazdą — programistką, która iście po mistrzowsku opanowała obszar swojej specjalizacji. Dostała awans na stanowisko kierownicze, co wiązało się ze wzrostem odpowiedzialności, i stanęła na wysokości zadania. Nie minęło wiele czasu, by stała się mentorką dla wszystkich wokół i zaczęła ich uczyć fachu. Zabierała głos na konferencjach poświęconych jej specjalizacji i wkrótce miała pod sobą kilka zespołów. Wprost uwielbiała być ekspertką. A jednak zaczęła się nudzić. Gdzieś po drodze przestała się uczyć nowych rzeczy. Novum bycia najmądrzejszą, najbardziej doświadczoną osobą na sali szybko straciło urok. Pomimo zewnętrznych oznak mistrzostwa i sukcesu czegoś jej brakowało. Któregoś dnia po przyjściu do pracy uświadomiła sobie, że jej dziedzina nie jest już tak istotna jak kiedyś, ludzie zajęli się innymi, ciekawymi sprawami. Gdzie popełniła błąd?

Spójrzmy prawdzie w oczy: *fajnie* jest być najmądrzejszą osobą na sali, a bycie mentorem dla innych bywa niezmiernie satysfakcjonujące. Problem polega na tym, że gdy osiągniesz szczyt możliwości w ramach swojego zespołu, przestajesz się uczyć. A kiedy przestajesz się uczyć, zaczynasz się nudzić. Albo nieoczekiwanie stajesz się zbędny. Naprawdę łatwo jest się uzależnić od bycia głównym rozgrywającym, ale trzeba wyrzec się odrobiny ego, aby móc obrać nowy kierunek i zapoznać się z czymś nowym. Znów chodzi o bycie bardziej *pokornym* i o to, by uczyć się równie chętnie co uczyć innych. Raz na jakiś czas wyjdź ze swojej strefy komfortu, znajdź staw, w którym pływają ryby grubsze od Ciebie, i zmierz się z wyzwaniami, jakie przed Tobą postawią. Na dłuższą metę będziesz dzięki temu o wiele szczęśliwszy.

Kup książkę Poleć książkę

Ucz się cierpliwości

Wiele lat temu Fitz pisał narzędzie konwertujące repozytoria CVS na Subversion (a potem na Gita) i wskutek kaprysów CVS nieustannie odkrywał jakieś dziwne błędy. Ponieważ jego długoletni przyjaciel i współpracownik Karl znał CVS od podszewki, postanowili podjąć współpracę na rzecz usunięcia tych błędów.

Problem pojawił się wówczas, gdy zaczęli programować razem. Fitz był przykładem inżyniera wyznającego wstępującą (*bottom-up*) metodykę postępowania. Nie wahał się skakać na głęboką wodę, błyskawicznie wypróbowywać wiele rozwiązań i wypływać na wierzch, bez zwracania większej uwagi na detale. Z kolei Karl był programistą działającym w myśl metody zstępującej (*top-bottom*), który zanim przystąpił do usuwania błędu, chciał najpierw zyskać pełny ogląd sytuacji i zagłębić się w implementację niemal wszystkich metod w stosie wywołań. Doprowadziło to do kilku kolosalnych konfliktów, nieporozumień, a nawet gorących sprzeczek. Doszło do tego, że panowie nie mogli programować w parze: było to dla nich zbyt irytujące.

Obaj od dawna mieli jednak do siebie zaufanie i szacunek. W połączeniu z cierpliwością pomogło im to wypracować nową metodę współpracy. Siadali przy komputerze, identyfikowali błąd, dzielili się i atakowali problem z dwóch stron jednocześnie (obiema metodami, wstępującą i zstępującą), a potem ponownie siadali razem i niejako spotykali się w połowie drogi, wymieniając się spostrzeżeniami. Ich cierpliwość i chęć do improwizowania z nowymi metodami pracy uratowała nie tylko projekt, ale też przyjaźń.

Bądź otwarty na wpływy

Im bardziej jesteś otwarty na wpływy, tym sam możesz skuteczniej wpływać; im bardziej się odsłaniasz, tym silniejszy się zdajesz. Te stwierdzenia mogą brzmieć jak dziwaczne oksymorony, chyba każdy jednak ma w pamięci kogoś, z kim pracował, a kto przejawiał iście ośli upór. Im usilniej ludzie starali się go przekonać, tym mocniej się opierał. Jaki los ostatecznie spotyka takich członków zespołu? Z naszego doświadczenia wynika, że zaczyna się ich „wymijać" jak przeszkody, do których istnienia wszyscy się przyzwyczaili. Ludzie przestają słuchać ich opinii albo zastrzeżeń. Na pewno nie chcesz, aby przydarzyło Ci się coś takiego, pamiętaj więc jedno: nie ma nic złego w tym, że pod czyimś wpływem zmienisz zdanie. Ostrożnie wybieraj swoje bitwy. Pamiętaj, że abyś został wysłuchany, najpierw musisz wysłuchać innych. Jeśli chodzi o wpływy, to wysłuchanie powinno mieć

Kup książkę Poleć książkę

miejsce, zanim jeszcze położy się coś na szali albo ostatecznie zadeklaruje podjęcie jakiejś decyzji — jeśli będziesz nieustannie zmieniać zdanie, ludzie uznają Cię za chwiejnego.

Kwestia odsłaniania się także początkowo wydaje się dziwna. Jeśli ktoś otwarcie przyznaje, że nie ma pojęcia na jakiś temat albo nie wie, jak rozwiązać problem, to jaką wiarygodnością będzie się cieszył w zespole? Czyż nie jest tak, że odsłanianie się jest przejawem słabości, a to niszczy zaufanie?

Nie jest. Przyznanie się do błędu albo do tego, że coś Cię przerasta, jest w dłuższej perspektywie sposobem na *poprawę* reputacji. Tak naprawdę działanie to obejmuje wszystkie aspekty HRT: jest to jawne okazanie *pokory*, dowód wiarygodności i brania na siebie odpowiedzialności i sygnał, że *ufasz* opiniom innych — w zamian zaś ludzie *szanują* Twoją szczerość i siłę. Czasami najlepsze, co można zrobić, to powiedzieć „nie wiem".

Kup książkę Poleć książkę

Weźmy choćby zawodowych polityków: ci ludzie słyną z tego, że nigdy nie potrafią przyznać się do błędu albo do ignorancji, nawet jeśli jest absolutnie oczywiste, że się mylą albo nie mają pojęcia na jakiś temat. Z tego względu większość ludzi nie wierzy ani jednemu słowu, które padło z ust polityków. Ta postawa wynika głównie z faktu, że politycy są pod nieustannym ostrzałem przeciwników. Podczas pisania programów nie trzeba jednak żyć w ciągłym poczuciu zagrożenia — koledzy z zespołu są współpracownikami, a nie rywalami.

KOLEJNE KROKI

Jeśli dotarłeś aż dotąd, to jesteś na dobrej drodze do mistrzowskiego opanowania sztuki pozostawania w dobrej komitywie z innymi. Zacznij od przyjrzenia się własnym zachowaniom i przemyślenia ich. Kiedy wdrożysz przedstawione strategie w codziennym życiu, przekonasz się, że współpraca stanie się o wiele naturalniejsza, a Twoja inżynieryjna produktywność zacznie zauważalnie wzrastać.

Ważne zmiany zaczynają się od Ciebie, a potem przenoszą się na innych. W następnym rozdziale będzie mowa o tworzeniu kultury HRT w Twoim najbliższym zespole.

Kup książkę Poleć książkę

Skorowidz

Kup książkę Poleć książkę

Kup książkę Poleć książkę

R

S

Ś

T

U

W

Kup książkę Poleć książkę

Z

Kup książkę Poleć książkę

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

Mniej czasu na kłótnie, więcej na współpracę!

Programiści i inżynierowie oprogramowania spędzają mnóstwo czasu na zgłębianiu tajników algorytmów, kodu czy technik budowy aplikacji. Bardzo mało uwagi jednak poświęcają takim zagadnieniom jak komunikacja, skuteczna praca w zespole czy prowadzenie wspólnych projektów. Tymczasem programowanie jest jednym z zadań, które realizuje się w grupie. Oznacza to, że sposób pracy zespołu ma bezpośredni i znaczący wpływ na produktywność i zadowolenie jego członków. Zaniedbanie tej kwestii było przyczyną niepowodzenia wielu obiecujących projektów.

Niniejsza książka, choć pozornie adresowana do liderów zespołów programistów, przyda się każdemu, kto bierze udział w dowolnym grupowym przedsięwzięciu o twórczym charakterze: począwszy od klubu studenckiego, skończywszy na korporacji architektów. Znalazły się tu praktyczne wskazówki dotyczące efektywnego kierowania zespołem, poruszania się w ramach organizacji oraz budowania zdrowych relacji z użytkownikami. Nie zabrakło porad z zakresu różnych sposobów komunikowania się i skutecznego wpływania na drugiego człowieka, a także metod radzenia sobie z trudnymi ludźmi. To znakomita lektura dla każdego twórczego człowieka!

W tej książce:

- wzorce komunikacyjne a kultura zespołu
- przywództwo: wzorce i antywzorce
- skuteczne strategie współpracy z innymi zespołami w ramach organizacji
- identyfikowanie i zażegnywanie zagrożeń związanych z komunikacją
- zarządzanie opinią publiczną

Brian Fitzpatrick przez kilka lat prowadził duże projekty inżynieryjne dla firmy Google, takie jak Data Liberation Front i Transparency Engineering. Przed przejściem do Google pracował m.in. w Apple i CollabNet. Jest autorem wielu artykułów. Mieszka w Chicago.

Ben Collins-Sussman tworzył system kontroli wersji Subversion. Od lat pracuje dla firmy Google, zainicjował projekt Google Code, zajmował się technologią wyświetlania reklam oraz infrastrukturą wyszukiwarki. W wolnych chwilach gra na banjo i komponuje musicale.

Helion

helion.pl

HELION SA
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl

Sprawdź nasze szkolenia!

SZKOLENIA

AKADEMIA IT & BUSINESS

WWW.SZKOLENIA.HELION.PL

KOD KORZYŚCI
Sięgnij po więcej!

ISBN 978-83-283-4338-2

9 788328 343382

INFORMATYKA W NAJLEPSZYM WYDANIU

Cena: 39,90 zł