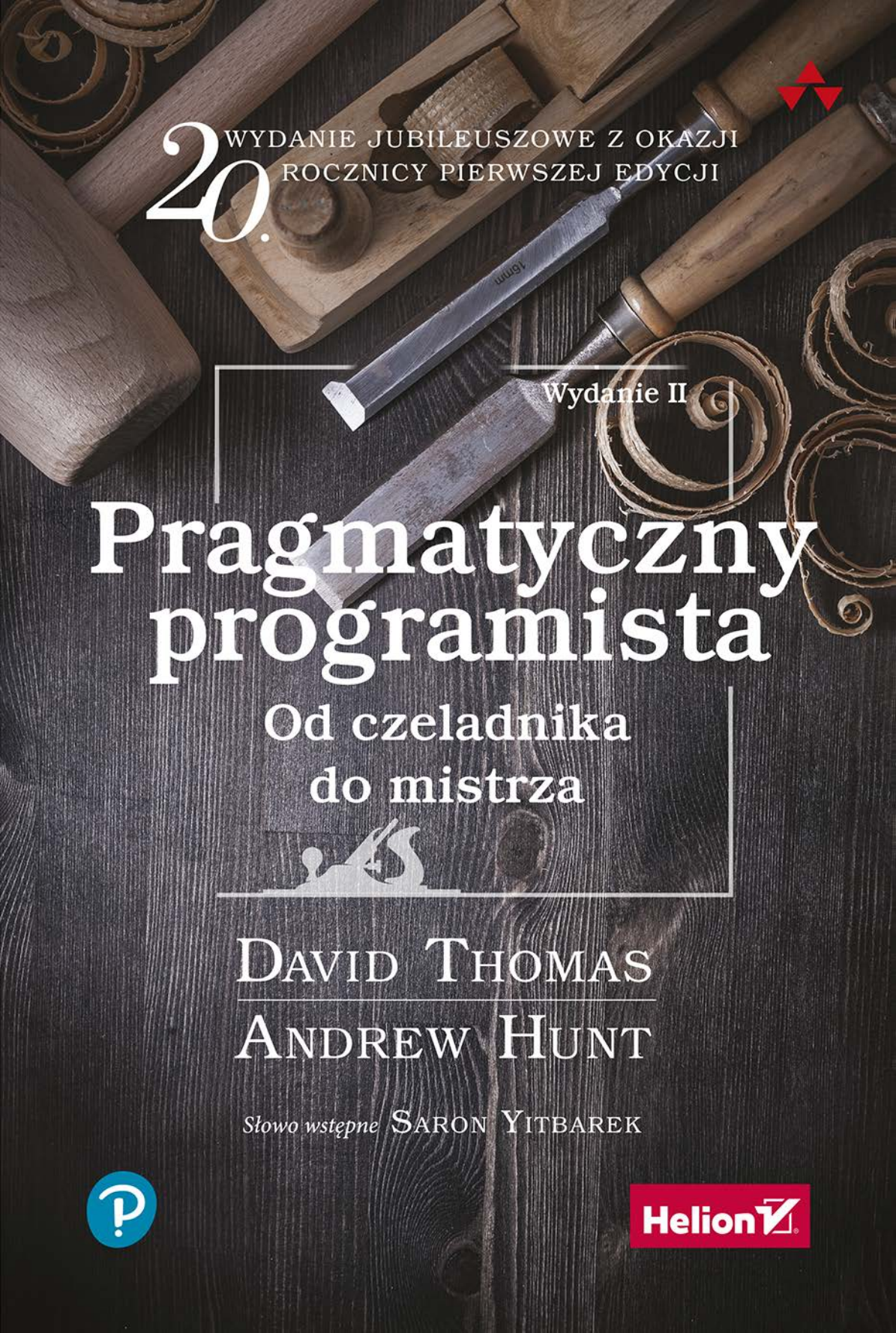
20.
WYDANIE JUBILEUSZOWE Z OKAZJI
ROCZNICY PIERWSZEJ EDYCJI
Wydanie II
Pragmatyczny programista
Od czeladnika do mistrza
DAVID THOMAS
ANDREW HUNT
Słowo wstępne SARON YITBAREK
Helion

Tytuł oryginału: The Pragmatic Programmer: Your Journey To Mastery, 20th Anniversary Edition (2nd Edition)

Tłumaczenie: Radosław Meryk
na podstawie Pragmatyczny programista. Od czeladnika do mistrza
w przekładzie Mikołaja Szczepaniaka

ISBN: 978-83-283-7139-2

Helion SA
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/pragp2
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

Rozdział 1.

Filozofia pragmatyczna

Ta książka jest o Tobie.

Nie popełnij błędu. Tu chodzi o Twoją karierę, a co ważniejsze, *to jest Twoje życie*. Należy do Ciebie. Czytasz tę książkę, bo wiesz, że możesz stać się lepszym programistą, a także pomagać innym, aby również stali się lepszymi. Możesz stać się pragmatycznym programistą.

Co wyróżnia pragmatycznych programistów? Czujemy, że pragmatyzm to postawa, styl, filozofia postrzegania i rozwiązywania problemów. Pragmatyczni programiści wykraczają myślami poza bieżące, aktualnie rozwiązywane problemy, stale próbując sytuować te problemy w szerszym kontekście, aby dysponować pełnym obrazem analizowanej rzeczywistości. Czy bez świadomości tego szerszego kontekstu w ogóle możemy być pragmatyczni? Jak w takiej sytuacji mielibyśmy wypracowywać inteligentne kompromisy i podejmować świadome decyzje?

Innym kluczem do sukcesu pragmatycznych programistów jest gotowość do brania odpowiedzialności za wszystko, co robią — to zagadnienie zostanie omówione w podrozdziale „Kot zjadł mój kod źródłowy". Odpowiedzialność oznacza, że pragmatyczni programiści nie siedzą bezczynnie, obserwując, jak ich projekty zmierzają ku nieuchronnej klęsce. W podrozdziale „Entropia oprogramowania" zostaną omówione sposoby dbania o nieskazitelność projektów.

Większość ludzi z trudem akceptuje zmiany — niechęć do zmian w pewnych przypadkach jest w pełni uzasadniona, ale nierzadko wynika ze zwykłego marazmu. W podrozdziale „Zupa z kamieni i gotowane żaby" przeanalizujemy strategię inspirowania zmian i przedstawimy (dla równowagi) pouczającą opowieść o płazie, który ignorował niebezpieczeństwa związane ze stopniowymi zmianami.

Jedną z korzyści wynikających ze znajomości i rozumienia kontekstu, w którym pracujemy, jest łatwiejsza ocena tego, na ile dobre musi być tworzone przez nas oprogramowanie. W pewnych przypadkach jedynym akceptowanym stanem jest jakość bliska perfekcji, ale często możliwe są daleko idące kompromisy.

Tym zagadnieniem zajmiemy się w podrozdziale „Odpowiednio dobre oprogramowanie".

Zapanowanie nad tym wszystkim wymaga, oczywiście, szerokiej wiedzy i sporego doświadczenia. Uczenie się jest typowym przykładem procesu ciągłego, który nigdy się nie kończy. W podrozdziale „Portfolio wiedzy" omówimy pewne strategie zachowywania właściwego tempa zdobywania wiedzy i umiejętności.

I wreszcie, nikt z nas nie pracuje w próżni. Wszyscy spędzamy znaczną część swojego czasu na interakcji ze współpracownikami. W podrozdziale „Komunikuj się!" zostaną omówione sposoby doskonalenia zasad współpracy.

Programowanie pragmatyczne rozciąga się od filozofii do myślenia pragmatycznego. W tym rozdziale skoncentrujemy się na podstawach filozofii.

To jest Twoje życie

Nie żyję w tym świecie, aby sprostać Waszym oczekiwaniom, a Wy nie żyjecie w nim po to, by sprostać moim.

Bruce Lee

To jest *Twoje* życie. Należy do Ciebie. Ty sam je przeżywasz. Ty sam je tworzysz.

Rozmawiamy z wieloma sfrustrowanymi programistami. Mają różne punkty widzenia. Niektórzy odczuwają, że w ich pracę wkradła się stagnacja, inni — że technologia przeszła obok nich. Ludzie czują się niedostatecznie doceniani albo niedostatecznie opłacani, albo że ich zespoły są toksyczne. Niektórzy chcieliby przenieść się do Azji lub Europy, albo pracować z domu.

Nasza odpowiedź jest zawsze taka sama.

„Dlaczego tego nie zmienisz?".

Wytwarzanie oprogramowania powinno znaleźć się blisko szczytu listy karier, nad którymi masz kontrolę. Nasze umiejętności są poszukiwane, nasza wiedza przekracza granice geograficzne, możemy pracować zdalnie. Jesteśmy dobrze opłacani. Naprawdę możemy zrobić niemal wszystko, czego chcemy.

Ale z jakiegoś powodu programiści są niechętni zmianom. Zamykają się w sobie w nadziei, że sytuacja się poprawi. Biernie godzą się na to, że ich umiejętności stają się przestarzałe i skarżą się, że ich firmy nie wysyłają ich na szkolenia. Jadąc autobusem przyglądają się ogłoszeniom z egzotycznych miejsc, a następnie wychodzą z autobusu na chłód i deszcz i brną do pracy.

Oto najważniejsza wskazówka w tej książce.

WSKAZÓWKA NR 3

Masz w sobie możliwości sprawcze.

Denerwuje Cię środowisko, w którym pracujesz? Twoja praca jest nudna? Spróbuj temu zaradzić. Ale nie próbuj w nieskończoność. Jak mówi Martin Fowler: „Można zmienić firmę lub zmienić swoją firmę”[1].

Jeśli wydaje Ci się, że technologia przechodzi obok Ciebie, znajdź czas (w swoim czasie wolnym) na studiowanie nowych zagadnień, które wydają się interesujące. Inwestujesz w siebie, więc robienie tego w czasie wolnym jest bardzo rozsądne.

Chcesz pracować zdalnie? A czy zapytałeś? Jeśli powiedzą Ci „nie”, to znajdź kogoś, kto powie Ci „tak”.

Ta branża daje bardzo wiele szans. Bądź proaktywny i z nich skorzystaj.

Pokrewne podrozdziały

- Temat 4., „Zupa z kamieni i gotowane żaby”.
- Temat 6., „Portfolio wiedzy”.

2 Kot zjadł mój kod źródłowy

Największą słabością jest strach przed wyglądaniem na słabego.

J.B. Bossuet, *Politics from Holy Writ*, 1709

Jednym z największych elementów filozofii pragmatycznej jest idea brania odpowiedzialności zarówno za siebie, jak i za skutki podejmowanych przez siebie działań (w wymiarze całej kariery, bieżącego projektu i codziennej pracy). Pragmatyczny programista bierze we własne ręce losy swojej kariery i nie boi się przyznać do braku wiedzy czy popełnionego błędu. Przyznawanie się do usterek z pewnością nie jest najprzyjemniejszym aspektem programowania, ale błędy są nieodłączną częścią tej pracy (nawet w najlepszych projektach). Błędy zdarzają się mimo gruntownych testów, dobrej dokumentacji i właściwie zaplanowanej automatyzacji. Dotrzymanie terminu bywa niewykonalne. Programiści napotykają nieprzewidywalne problemy techniczne.

Wspomniane zjawiska po prostu się zdarzają, a rolą programistów jest możliwie profesjonalne radzenie sobie w trudnych sytuacjach. W tym przypadku profesjonalizm wymaga uczciwego i bezpośredniego stawiania sprawy. Mimo dumy ze swoich umiejętności musimy mieć odwagę uczciwego przyznawania się do słabszych stron, w tym braku wiedzy oraz popełnianych błędów.

[1] *http://wiki.c2.com/?ChangeYourOrganization*

Zaufanie zespołu

Przede wszystkim Twój zespół musi być w stanie Ci zaufać i na Tobie polegać. Ty także powinieneś móc polegać na każdym członku Twojego zespołu. Według literatury naukowej[2] zaufanie w zespole jest absolutnie niezbędne dla zapewnienia kreatywności i współpracy. W zdrowym, opartym na zaufaniu środowisku, można bezpiecznie mówić to, co się myśli, przedstawiać swoje pomysły i polegać na członkach zespołu, którzy mogą z kolei polegać na Tobie. Co by było, gdyby nie było zaufania? Cóż...

Wyobraźmy sobie, że tajny zespół high-tech ninja pracuje zawzięcie nad infiltracją siedziby złoczyńcy. Po miesiącach planowania i prób uruchamiania kodu wreszcie wszystko jest gotowe. Teraz Twoja kolej, aby skonfigurować siatkę naprowadzania laserowego. Mówisz: „Przepraszam was, ale nie mam lasera. Kot bawił się czerwoną kropką i zostawiłem laser w domu".

Tego rodzaju nadużycie zaufania może być trudne do naprawienia.

Bierz odpowiedzialność

Odpowiedzialność to cecha powszechnie uznawana za pożądaną. Mimo zaangażowania i dbałości o możliwie najlepszą realizację zadania nie zawsze mamy pełną kontrolę nad wszystkimi aspektami naszej pracy. Oprócz jak najlepszego wykonywania własnych działań musimy więc analizować sytuację pod kątem czynników ryzyka wykraczających poza naszą kontrolę. Odrzucenie odpowiedzialności jest usprawiedliwione tylko wtedy, gdy sytuacja uniemożliwia nam prawidłową realizację zadania lub gdy czynniki ryzyka są zbyt poważne. Ocena należy do samego programisty i zależy od jego zasad etycznych i osądu sytuacji.

Jeśli *bierzemy* na siebie odpowiedzialność za efekt podejmowanych działań, musimy być przygotowani na wszelkie konsekwencje. W razie popełnienia błędu (a wszyscy je popełniamy) lub błędnej oceny sytuacji musimy uczciwie przyznać się do porażki i zaproponować rozwiązanie.

Nie należy winić za własne niedociągnięcia współpracowników ani innych czynników. Nie powinniśmy też szukać usprawiedliwień. Nie należy zrzucać winy za wszystkie problemy na producentów narzędzi, język programowania, przełożonych ani współpracowników. Każdy z tych elementów mógł, oczywiście, przyczynić się do powstałej sytuacji, jednak rolą programisty nie jest szukanie usprawiedliwień, tylko tworzenie rozwiązań.

Jeśli istnieje ryzyko niedostarczenia niezbędnych składników przez kogoś innego, należy przygotować odpowiedni plan awaryjny. Jeśli wskutek awarii dysku programista traci cały kod źródłowy i jeśli nie dysponuje kopią zapasową, wina

[2] Wartościową metaanalizę dotyczącą zaufania i wydajności zespołów można znaleźć w artykule: *A meta-analysis of main effects, moderators, and covariates*, *http://dx.doi.org/10.1037/apl0000110*.

leży wyłącznie po jego stronie. Problemu nie da się rozwiązać, wmawiając szefowi, że kod źródłowy został zjedzony przez kota.

WSKAZÓWKA NR 4

Proponuj rozwiązania, zamiast posługiwać się kiepskimi wymówkami.

Zanim udamy się do kogokolwiek, aby tłumaczyć, dlaczego wykonanie jakiejś czynności jest niemożliwe, dlaczego nie możemy dotrzymać terminu lub dlaczego coś nie spełnia początkowych wymagań, warto zatrzymać się na chwilę i wsłuchać się we własne wyjaśnienia. Warto porozmawiać z gumową kaczką na monitorze, kotem lub czymkolwiek innym. Czy te wyjaśnienia brzmią logicznie, czy po prostu głupio? Jak będą brzmiały dla naszego przełożonego?

Warto przeprowadzić tę rozmowę we własnej głowie. Jakiej odpowiedzi spodziewamy się po rozmówcy? Czy zapyta: „Próbowałeś tego? Nie pomyślałeś o tym”? Jak wtedy odpowiemy? Czy możemy zrobić coś jeszcze, zanim udamy się do przełożonego ze złymi nowinami? W pewnych przypadkach z góry wiadomo, co powie przełożony, zatem warto oszczędzić mu zmartwień.

Zamiast usprawiedliwień należy raczej przygotować propozycje rozwiązań. Nie należy mówić o tym, czego nie można zrobić — powinniśmy raczej koncentrować się na tym, co zrobić, aby wyjść z kłopotliwej sytuacji. Czy porzucenie dotychczasowego kodu rzeczywiście jest konieczne? Warto przedstawić rozmówcy zalety refaktoryzacji (patrz temat 40., „Refaktoryzacja”).

Czy planujemy poświęcić czas na przygotowanie prototypu, który ułatwi nam wypracowanie najlepszego rozwiązania (patrz temat 13., „Prototypy i karteczki samoprzylepne”)? Czy w przyszłości będzie można uniknąć podobnych sytuacji, jeśli zostaną wdrożone lepsze procedury testowania (patrz temat 41., „Kod łatwy do testowania” i podrozdział „Bezlitosne testy”) lub rozwiązania w zakresie automatyzacji.

Być może właściwym rozwiązaniem będzie zapewnienie dodatkowych zasobów. A może trzeba poświęcić więcej czasu użytkownikom? A może wszystko leży w Twoich rękach: być może powinieneś dokładniej nauczyć się jakiejś techniki lub technologii? Może trzeba sięgnąć po książkę lub zapisać się na kurs? Programista nie powinien obawiać się zadawania pytań ani przyznawania do tego, że potrzebuje pomocy.

Warto próbować eliminować kiepskie wymówki jeszcze przed ich głośnym wypowiedzeniem. A jeśli odczuwamy nieodpartą potrzebę ich wyrażenia, przedstawmy te usprawiedliwienia raczej swojemu kotu. Skoro mały Mruczek jest skłonny wziąć winę na siebie, dlaczego nie skorzystać z tej okazji…

Pokrewne podrozdziały

- Temat 49., „Pragmatyczne zespoły”.

Wyzwania

- Jak reagujemy na kiepskie wymówki stosowane przez innych (pracowników banku, mechaników samochodowych, urzędników itp.) podczas rozmowy z nami? Jaki jest wpływ tych wymówek na naszą ocenę rozmówców i organizacji, w których pracują?
- Gdy kiedykolwiek znajdziesz się w sytuacji, że będziesz zmuszony odpowiedzieć: „Nie wiem", pamiętaj, aby dodać „ale się dowiem". To świetny sposób, aby się przyznać, że czegoś się nie wie, a następnie by przyjąć na siebie odpowiedzialność, tak jak powinien postąpić profesjonalista.

3 Entropia oprogramowania

Chociaż wytwarzanie oprogramowania jest odporne na niemal wszystkie prawa fizyki, akurat zjawisko entropii jest dla programistów aż nadto odczuwalne. **Entropia** to termin zaczerpnięty z fizyki i opisujący stopień nieokreśloności, chaotyczności w systemie. Prawa termodynamiki mówią, niestety, że entropia we wszechświecie zmierza do osiągnięcia pewnego maksimum. Kiedy zjawisko chaosu zaczyna nasilać się w świecie oprogramowania, mamy do czynienia ze zjawiskiem określanym mianem rozkładu oprogramowania (ang. *software rot*). Niektórzy to samo zjawisko określają bardziej optymistycznym terminem „dług techniczny", z domniemanym założeniem, że pewnego dnia go spłacą. Prawdopodobnie im się to nie uda.

Niezależnie od nazwy, zarówno dług techniczny, jak i rozkład oprogramowania, mogą nasilać się w niekontrolowany sposób.

Istnieje wiele czynników, które mogą się składać na rozkład oprogramowania. Bodaj najważniejszym czynnikiem są cechy psychologiczne (lub kulturowe) osób zaangażowanych w realizację projektu. Nawet w przypadku jednoosobowego „zespołu" psychologia projektu może okazać się bardzo delikatna. Nawet najlepsze plany i najlepsi ludzie nie wystarczą do zabezpieczenia projektu przed ruiną i upadkiem w całym czasie życia. Co ciekawe, istnieją projekty, które wbrew licznym utrudnieniom i mimo pasma niepowodzeń zadziwiająco skutecznie radzą sobie z naturalnym dążeniem do nieporządku i doskonale znoszą upływ czasu.

Co decyduje o tej różnicy?

W centrach miast tylko część budynków jest pięknych i zadbanych, podczas gdy pozostałe to rozpadające się rudery. Dlaczego? Badacze zjawiska przestępczości i degradacji życia społecznego w miastach odkryli bardzo ciekawy mechanizm, który może sprawić, że czysty, zadbany, zamieszkany budynek błyskawicznie pustoszeje i zaczyna niszczeć[3].

[3] Patrz *The police and neighborhood safety* [WH82].

Wystarczy jedna wybita szyba.

Jedna wybita szyba pozostawiona nienaprawiona przez dłuższy czas powoduje, że mieszkańcy zaczynają traktować swój dom jako opuszczone, porzucone miejsce, które nie może liczyć na właściwą opiekę wyznaczonych do tego podmiotów. Niedługo potem zostaje zbita kolejna szyba. Ludzie zaczynają pozostawiać śmieci w przypadkowych miejscach. Na murach pojawia się graffiti. Po jakimś czasie budynek jest już narażony na poważne uszkodzenia strukturalne. W stosunkowo krótkim okresie budynek znajduje się w stanie, w którym przywrócenie dawnej świetności traci sens ekonomiczny, zatem pierwotne wrażenie porzucenia staje się rzeczywistością.

Dlaczego ta jedna wybita szyba stwarza tak wielką różnicę? Z badań psychologów[4] wynika, że pokazanie stanu beznadziejności może być zaraźliwe. Weźmy za przykład rozprzestrzenianie się wirusa grypy w dużych skupiskach ludzkich. Ignorowanie wyraźnie nieprawidłowej sytuacji wzmacnia pogląd, że prawdopodobnie *nic* nie można naprawić, że nikogo to nie interesuje, wszystko jest przesądzone. Pojawiają się negatywne myśli, które mogą rozprzestrzeniać się między członkami zespołu, tworząc błędne koło.

WSKAZÓWKA NR 5

Nie akceptuj żadnej wybitej szyby.

Nie należy pozostawiać nienaprawionej żadnej wybitej szyby (złego projektu, niewłaściwej decyzji, kiepskiego kodu itp.). Każdą taką „szybę" należy wymienić zaraz po odkryciu problemu. Jeśli brakuje czasu na wstawienie nowej szyby, należy *wstawić w jej miejsce choćby dyktę*. W większości przypadków można sobie pozwolić na umieszczenie problematycznego kodu w komentarzu, wyświetlenie komunikatu „Do zaimplementowania" lub celowe wstawienie błędnych danych, które na pewno zwrócą uwagę programistów. Należy też podjąć działania nie tylko na rzecz ograniczenia ryzyka wywołania dalszych szkód przez wykrytą usterkę, ale też w celu dowiedzenia, że panujemy nad sytuacją.

Każdy z nas widział piękne, w pełni funkcjonalne systemy, które rozsypywały się jak domek z kart, kiedy tylko pojawiły się „wybite szyby". Istnieją też inne czynniki prowadzące do rozkładu oprogramowania (wrócimy do nich w dalszej części tej książki), ale nic tak nie *przyspiesza* tego zjawiska jak zaniedbania.

Część programistów sądzi, że konsekwentne sprzątanie rozbitego szkła i wstawianie nowych szyb jest niewykonalne — że nikt nie ma na to czasu podczas realizacji projektu. Programiści, którzy podzielają ten pogląd, powinni albo zaopatrzyć się w duży kubeł na śmieci, albo zmienić sąsiedztwo. Nie pozwólmy entropii wygrać.

[4] Patrz *Contagious depression: Existence, specificity to depressed symptoms, and the role of reassurance seeking* [Joi94].

Po pierwsze nie szkodzić

Andy miał kiedyś znajomego, który był nieprzyzwoicie bogaty. Jego dom był pełen bezcennych antyków, dzieł sztuki itp. Pewnego dnia zapalił się gobelin zawieszony zbyt blisko kominka w salonie. Wezwano na pomoc straż pożarną, aby zapobiec katastrofie. Mimo szalejącego pożaru strażacy poprzedzili rozwinięcie swoich wielkich, brudnych węży starannym rozłożeniem mat pomiędzy drzwiami wejściowymi a źródłem ognia.

W ten sposób zabezpieczyli dywan przed niepotrzebnym zabrudzeniem.

Opisany scenariusz z pewnością jest mocno przesadzony. Zapewne najważniejszym priorytetem straży pożarnej powinno być ugaszenie pożaru, bez względu na inne szkody. Jednak straż pożarna najwyraźniej właściwie oceniła sytuację. Strażacy byli przekonani o zdolności do zapanowania nad ogniem i zastosowali środki ostrożności w celu uniknięcia niepotrzebnych zniszczeń w budynku. W podobny sposób należy postępować z oprogramowaniem: nie należy powodować dodatkowych szkód tylko dlatego, że powstał jakiś kryzys. Jedna wybita szyba to o jedna za dużo.

Jedna wybita szyba (źle zaprojektowany fragment kodu, błędna decyzja kierownictwa utrudniająca pracę zespołu w całym okresie realizacji projektu itp.) w zupełności wystarczy do rozpoczęcia procesu degradacji. Programista zaangażowany w projekt, w którym aż roi się od takich „wybitych szyb", nie może pogodzić się z sytuacją i przyjąć postawy: „Skoro cała reszta kodu nadaje się do kosza, po co mój kod miałby się czymś wyróżniać?". Jakość projektu do momentu, w którym trafia do rąk programisty, nie powinna mieć żadnego znaczenia. W oryginalnym projekcie, który doprowadził do powstania teorii wybitej szyby, porzucony samochód początkowo stał nietknięty przez tydzień. Wystarczyło jednak wybicie jednej szyby, aby w ciągu zaledwie paru *godzin* samochód został rozebrany niemal do gołej karoserii.

Podobnie, programista dołączający do zespołu realizującego projekt, którego kod jest przykładem nieskazitelnego piękna (jest wyjątkowo przejrzysty, doskonale zaprojektowany i elegancki), najprawdopodobniej będzie robił wszystko, aby niczego nie zepsuć (jak strażacy w przytoczonej anegdocie). Nawet w warunkach szalejącego pożaru (zbliżającego się terminu, nadchodzącej daty wydania, planowanej prezentacji itp.) *nikt* nie chce być pierwszą osobą, która zakłóci dotychczasową harmonię.

Po prostu sobie powiedz: „nie ma przyzwolenia na wybite szyby".

Pokrewne podrozdziały

- Temat 10., „Ortogonalność".
- Temat 40., „Refaktoryzacja".
- Temat 44., „Nazewnictwo".

Wyzwania

- Warto pomóc we wzmocnieniu swojego zespołu poprzez dokonanie gruntownego przeglądu otoczenia, w którym pracuje. Należy wybrać dwie lub trzy „wybite szyby", po czym omówić ze współpracownikami źródła poszczególnych problemów i możliwe rozwiązania.
- Czy potrafimy wskazać szybę, która została wybita jako pierwsza? Jaka była nasza reakcja po jej pierwszym wykryciu? Jeśli ta „wybita szyba" wynika z decyzji kogoś innego lub polecenia kierownictwa, jak możemy temu zaradzić?

4

Zupa z kamieni i gotowane żaby

Trzej głodni żołnierze wracali z wojny. Kiedy zobaczyli na swojej drodze wioskę, byli pełni nadziei — byli przekonani, że mieszkańcy osady zaproszą ich na posiłek. Po dotarciu na miejsce odkryli jednak, że wszystkie drzwi i okna są pozamykane. Po wieloletniej wojnie mieszkańcy wsi dysponowali niewielką ilością jedzenia i pilnie strzegli swoich zapasów.

Zdeterminowani żołnierze zagotowali kocioł wody i ostrożnie włożyli do wrzątku trzy kamienie. Zaskoczeni wieśniacy zaczęli wychodzić ze swoich domów i obserwować poczynania żołnierzy.

— To jest zupa z kamieni — wyjaśnili żołnierze.

— To wszystko, co do niej włożycie? — dopytywali mieszkańcy osady.

— Oczywiście, chociaż są tacy, którzy twierdzą, że zupa jest jeszcze lepsza z kilkoma marchewkami. — Jeden z wieśniaków natychmiast pobiegł do swojego domu, by po chwili przynieść kosz pełen marchwi.

Kilka minut później wieśniacy znowu zaczęli dopytywać:

— To naprawdę wszystko?

— Cóż — odpowiedzieli żołnierze — kilka ziemniaków na pewno by nie zaszkodziło. — Po chwili inny wieśniak przybiegł z workiem ziemniaków.

Po godzinie żołnierze dysponowali składnikami, które w zupełności wystarczyły do przyrządzenia wymarzonej zupy: wołowiną, porem, solą i ziołami. Za każdym razem inny mieszkaniec wsi plądrował własną spiżarnię w poszukiwaniu składnika wskazanego przez żołnierzy.

Ostatecznie udało się ugotować całkiem spory kocioł doskonałej zupy. Żołnierze wyjęli z zupy kamienie, po czym usiedli wraz z mieszkańcami wsi, aby wspólnie celebrować pierwszy pełnowartościowy posiłek od wielu miesięcy.

Z historii o zupie z kamieni płynie wiele wniosków. Wieśniacy zostali oszukani przez żołnierzy, którzy wykorzystali ich ciekawość do zdobycia niezbędnych składników. Ważniejsze jest jednak coś zupełnie innego — żołnierze zadziałali jak katalizator skupiający społeczność wokół jednego celu i umożliwiający ugotowanie posiłku, którego poszczególni mieszkańcy wsi nie mogliby sporządzić na bazie własnych zapasów. Mamy więc do czynienia z pożądanym skutkiem synergii. Ostatecznie wszyscy odnieśli korzyści.

Warto więc spróbować zastosować metodę tych żołnierzy we własnym środowisku.

Możemy znaleźć się w sytuacji, w której doskonale wiemy, co należy zrobić i jak to osiągnąć. Wyobraźmy sobie, że mamy w głowie kompletną koncepcję systemu, o której z całą pewnością wiemy, że jest słuszna. Przyjmijmy jednak, że musimy uzyskać zgodę na to przedsięwzięcie i że spotykamy się z niechęcią i brakiem zrozumienia. Musimy przekonać rozmaite komitety, zadbać o akceptację zaproponowanego budżetu — projekt już na tym etapie bardzo się komplikuje. Każdy zazdrośnie strzeże swoich zasobów. Opisane zjawisko bywa określane mianem próby sił na początku działalności (ang. *start-up fatigue*).

To dobry moment, by sięgnąć po kamienie. Należy dobrze przemyśleć, o które składniki warto prosić. Warto dobrze wykorzystać każdy otrzymany składnik. Po otrzymaniu półproduktu należy zaprezentować efekt wszystkim zainteresowanym i pozwolić, by zachwycali się otrzymanym dziełem. Kiedy powiedzą: „*byłoby* jeszcze lepsze, gdyby dodać…”, można udawać, że proponowane ulepszenia są nieistotne. Wystarczy teraz rozsiąść się wygodnie i czekać, aż sami zaczną sugerować dodawanie funkcji, które od początku chcieliśmy zaimplementować w tym systemie. Wielu ludziom łatwiej przychodzi dołączanie do udanego projektu niż praca na niepewny sukces od podstaw. Warto więc tak opisać perspektywy projektu, aby byli przekonani o jego świetlanej przyszłości[5].

WSKAZÓWKA NR 6

Bądź katalizatorem zmian.

Mieszkańcy wsi

Jeśli spojrzymy na historię zupy z kamieni z punktu widzenia wieśniaków, otrzymamy opowieść o subtelnym i postępującym oszustwie. To historia o zbyt wąskiej perspektywie. Mieszkańcy osady są tak zaintrygowani kamieniami, że zapominają o bożym świecie. To samo zdarza się każdemu, codziennie. Pewne rzeczy potrafią zadziwiająco skutecznie przykuwać naszą uwagę.

[5] Pewnym pokrzepieniem mogą być słowa przypisywane kontradmirał dr Grace Hopper: „Łatwiej prosić o wybaczenie, niż uzyskać pozwolenie”.

Opisane symptomy są nam doskonale znane. Projekty powoli, ale nieuchronnie wymykają nam się z rąk. Większość katastrof w świecie oprogramowania rozpoczyna się od zjawisk tak niepozornych, że wręcz trudnych do dostrzeżenia. Co więcej, większość projektów nie jest realizowana w terminie. Systemy oddalają się od swoich oryginalnych specyfikacji funkcja po funkcji. Co więcej, kolejne łatki eliminujące usterki powodują, że z czasem produkt nie ma nic wspólnego z oryginałem. Na spadek morale i rozpad zespołu często składa się wiele drobnych zjawisk, tyle że kumulowanych w dłuższym okresie.

WSKAZÓWKA NR

Pamiętaj o szerszym kontekście.

Nigdy tego nie próbowaliśmy. Naprawdę. Mówi się, że żywa żaba wrzucona do gotującej się wody natychmiast wyskoczy z garnka. Jeśli jednak wrzucimy żabę do zimnej wody i zaczniemy tę wodę stopniowo nagrzewać, żaba nie zwróci uwagi na powolny wzrost temperatury i pozostanie w garnku, aż zostanie ugotowana.

Łatwo zauważyć, że w przypadku żaby mamy do czynienia z zupełnie innym problemem od tego, który omówiono we wcześniejszym podrozdziale na przykładzie zbitej szyby. Zgodnie z teorią wybitej szyby ludzie tracą zainteresowanie walką z entropią, kiedy odkrywają, że byliby w tej walce osamotnieni — że otaczająca ich społeczność bagatelizuje problem. Żaba nawet nie dostrzega zmiany.

Nie możemy upodabniać się do tej żaby. Musimy mieć na uwadze szerszy obraz sytuacji. Powinniśmy stale obserwować to, co dzieje się w otaczającym nas świecie, zamiast koncentrować się tylko na swoich zadaniach.

Pokrewne podrozdziały

- Temat 1., „To jest Twoje życie”.
- Temat 38., „Programowanie przez koincydencję”.

Wyzwania

- Podczas recenzowania wstępnej wersji tej książki John Lakos zwrócił uwagę na ciekawe zjawisko. Żołnierze stopniowo oszukiwali mieszkańców wioski, ale powodowana przez to oszustwo zmiana postawy wieśniaków była dla wszystkich korzystna. Stopniowe oszukiwanie żaby ostatecznie prowadzi do jej skrzywdzenia. Czy potrafimy stwierdzić, kiedy przyspieszanie zmian odbywa się metodą zupy z kamieni, a kiedy zupy z żaby? Czy ta ocena będzie subiektywna, czy obiektywna?
- Odpowiedz szybko, bez spoglądania, ile lamp jest na suficie nad Tobą? Ile jest ich wszystkich w pokoju? Ile osób w nim przebywa? Czy jest coś spoza kontekstu — coś, co wygląda jakby nie należało do wystroju? Jest to ćwiczenie na *świadomość sytuacyjną* — technikę praktykowaną zarówno

przez skautów, jak i oddziały marynarki wojennej Stanów Zjednoczonych. Nabierz nawyku obserwowania otoczenia i spostrzegania tego, co się w nim znajduje. Następnie stosuj ten sam nawyk w swoim projekcie.

5 Odpowiednio dobre oprogramowanie

Kto lepsze goni, często w gorsze wpadnie.

***Król Lear*, akt I, scena IV**

Istnieje stary dowcip o amerykańskiej firmie, która zamówiła 100 tysięcy układów scalonych u japońskiego producenta. Specyfikacja zawierała błąd, który powodował usterkę w jednym na 10 tysięcy układów. Kilka tygodni po złożeniu zamówienia do firmy dotarło wielkie pudło z tysiącami układów scalonych oraz małe pudełko z zaledwie dziesięcioma układami. W pudełku znajdowała się kartka z napisem: „Tutaj są te układy z błędami".

Możemy, oczywiście, tylko pomarzyć o podobnej kontroli nad jakością swoich produktów. Świat nigdy nie pozwoli nam tworzyć naprawdę doskonałego, w pełni wolnego oprogramowania. Przeciwko nam sprzysięgły się takie siły jak czas, technologia i temperament.

Nasza sytuacja wcale nie musi być frustrująca. Jak napisał Ed Yourdon w swoim artykule w „IEEE Software", *When good-enough software is best* [You95], przy odrobinie samodyscypliny możemy pisać oprogramowanie, które będzie wystarczająco dobre — wystarczająco dobre dla użytkowników, dla osób w przyszłości odpowiedzialnych za jego konserwację i dla naszego świętego spokoju. Szybko odkryjemy, że taki model pracy zapewnia nam maksymalną produktywność i w pełni satysfakcjonuje użytkowników. Co więcej, z zadowoleniem stwierdzimy, że nasze programy są lepsze także dzięki skróconemu procesowi produkcji.

Zanim przystąpimy do dalszych rozważań, musimy doprecyzować, co naprawdę oznaczają te propozycje. Określenie „wystarczająco dobre" nie oznacza niechlujnego czy źle opracowanego kodu. W przypadku każdego systemu warunkiem sukcesu jest zgodność z wymaganiami użytkowników. W tym podrozdziale proponujemy koncepcję, w której użytkownicy końcowi mają szansę udziału w procesie podejmowania decyzji o tym, kiedy nasze dzieło jest wystarczająco dobre.

Angażowanie użytkowników w rozstrzyganie o jakości

W typowych okolicznościach piszemy oprogramowanie z myślą o innych użytkownikach. Często pamiętamy nawet o potrzebie uzyskania wymagań od tych użytkowników[6]. Jak często pytamy użytkowników, *na ile dobrego* oprogramo-

[6] To miał być żart!

wania oczekują? W pewnych przypadkach taki wybór oczywiście jest niemożliwy. Jeśli pracujemy nad rozrusznikiem serca, promem kosmicznym lub niskopoziomową biblioteką używaną później przez tysiące innych programistów, wymagania będą bardziej restrykcyjne, a nasze pole manewru stosunkowo niewielkie.

Jeśli jednak pracujemy nad zupełnie nowym produktem, podlegamy całkiem innym ograniczeniom. Pracownicy działu marketingu składają przyszłym użytkownikom pewne obietnice, a sami użytkownicy mogą mieć plany związane z zapowiedzianym terminem wydania. Co więcej, nasza firma na pewno podlega ograniczeniom związanym z przepływem środków finansowych. Ignorowanie wymagań użytkowników w zakresie implementacji nowych funkcji lub tylko nieznacznego udoskonalenia gotowego kodu byłoby dalece nieprofesjonalne. Nie oznacza to jednak, że należy panicznie bać się negatywnej oceny użytkownika końcowego — równie nieprofesjonalne jest składanie nierealnych obietnic co do terminu i rezygnowanie z kolejnych elementów produktu wyłącznie z myślą o dotrzymaniu tych nieprzemyślanych deklaracji.

Zakres i jakość tworzonego systemu powinny być precyzyjnie określone w wymaganiach tego systemu.

WSKAZÓWKA NR 8

Jakość powinna być uwzględniona w wymaganiach.

W wielu przypadkach nie da się uniknąć wyboru części funkcji kosztem innych. Co ciekawe, wielu użytkowników jest skłonnych zgodzić się na korzystanie z okrojonej wersji *już dzisiaj*, zamiast czekać na przykład rok na wersję uzupełnioną o elementy multimedialne. Wiele działów IT dysponujących skromnym budżetem zapewne się z tym zgodzi. Dobre oprogramowanie dzisiaj często jest lepsze niż doskonałe oprogramowanie jutro. Jeśli z odpowiednim wyprzedzeniem przekażemy użytkownikom produkt, z którym będą mogli swobodnie eksperymentować, ich opinie i wskazówki najprawdopodobniej umożliwią nam wypracowanie lepszego produktu docelowego (patrz temat 12., „Pociski smugowe").

Warto wiedzieć, kiedy przestać

Programowanie pod pewnymi względami przypomina malowanie. W obu przypadkach praca rozpoczyna się od pustego płótna i kilku prostych materiałów. Dopiero połączenie nauki, sztuki i rzemiosła pozwala prawidłowo używać dostępnych środków. Należy zacząć od naszkicowania ogólnego kształtu, by następnie namalować otoczenie i wreszcie wypełnić szczegóły. Powinniśmy też stale odchodzić parę kroków od obrazu, by z większej odległości spojrzeć krytycznym okiem na dotychczasowe dokonania. Od czasu do czasu musimy nawet wyrzucić całe płótno do kosza i zacząć wszystko od nowa.

Każdy artysta potwierdzi jednak, że cała ta ciężka praca nie ma najmniejszego sensu, jeśli twórca nie wie, kiedy przestać. Jeśli bez końca nanosimy kolejne warstwy i domalowujemy kolejne szczegóły, *artystyczna wartość obrazu ginie gdzieś pod nadmiarem farby*.

Nie należy niszczyć dobrego programu przesadną liczbą upiększeń i wyrafinowanych dodatków. Powinniśmy raczej pozwolić, by nasz kod sprawdził się w działaniu bez naszego udziału. Być może nie jest doskonały. Nie ma jednak powodu do zmartwień — i tak nigdy taki by nie był. (W rozdziale 7. „Kiedy kodujemy…" omówimy filozofie tworzenia kodu w niedoskonałym świecie).

Pokrewne podrozdziały

- Temat 45., „Kopalnia wymagań".
- Temat 46., „Rozwiązywanie niemożliwych do rozwiązania łamigłówek".

Wyzwania

- Warto przyjrzeć się producentom narzędzi i systemów operacyjnych, których sami używamy. Czy potrafimy wskazać dowody sugerujące, że te firmy oferują oprogramowanie, o którym same wiedzą, że jest niedoskonałe? Czy jako użytkownicy wolelibyśmy raczej (1) czekać na wydanie oprogramowania pozbawionego wszystkich błędów, (2) otrzymać złożone oprogramowanie przy akceptacji pewnych niedoróbek, czy (3) korzystać z prostszego oprogramowania z nieco większą liczbą usterek?
- Zastanówmy się nad skutkami ewentualnej modularyzacji procesu dostarczania oprogramowania. Czy uzyskanie monolitycznego bloku oprogramowania w określonej jakości zajęłoby więcej, czy mniej czasu niż w przypadku projektowania systemu w formie zbioru modułów? Czy potrafimy wskazać jakieś komercyjne przykłady?
- Czy potrafisz wskazać popularne oprogramowanie, którego wadą jest *nadmiar funkcjonalności*? Oznacza to oprogramowanie zawierające znacznie więcej funkcji niż kiedykolwiek będziemy używać. Każda funkcjonalność wprowadza więcej możliwości popełnienia błędów i luk w zabezpieczeniach i sprawia, że trudniej jest znaleźć to, czego potrzebujemy, a także zarządzać funkcjonalnościami. Czy istnieje niebezpieczeństwo, że sam wpadniesz w tego rodzaju pułapkę?

6 Portfolio wiedzy

Inwestycja w wiedzę zawsze przynosi największe zyski.

Benjamin Franklin

Jak widać, zawsze można liczyć na celne i zwięzłe wskazówki starego, dobrego Bena Franklina. Czy do zostania doskonałymi programistami wystarczy wczesne chodzenie spać i wstawanie o świcie? Ranny ptaszek może, oczywiście, pierwszy dopaść dorodnego robaka, ale co na rannym wstawaniu zyskuje robak?

W tym przypadku Ben trafił w sedno. Wiedza i doświadczenie to zdecydowanie najważniejsze atuty w naszej profesji.

Okazuje się jednak, że wymienione *aktywa nie są wieczne*[7]. Nasza wiedza dezaktualizuje się wraz z powstawaniem nowych technik, języków i środowisk. Zmieniające się warunki rynkowe mogą powodować, że nasze dotychczasowe doświadczenia stają się wręcz bezwartościowe. Zważywszy na tempo zmian w erze internetu, opisane zjawiska mogą zachodzić wyjątkowo szybko.

Wraz ze spadkiem wartości naszej wiedzy spada wartość nas samych z perspektywy pracodawcy lub klienta. Naturalnym rozwiązaniem jest więc dążenie do zapobieżenia temu spadkowi.

Twoja zdolność do uczenia się nowych rzeczy, to najważniejszy zasób o strategicznym znaczeniu. Ale w jaki sposób dowiedzieć się, *jak* się uczyć oraz skąd wiedzieć, *czego* się uczyć?

Portfolio wiedzy

Wszyscy lubimy postrzegać wszystkie znane programiście fakty o przetwarzaniu komputerowym, wszystkie dziedziny, w których pracował ten programista, oraz jego łączne doświadczenie jako tzw. **portfolio wiedzy** programisty. Zarządzanie portfolio wiedzy pod wieloma względami przypomina zarządzanie portfelem instrumentów finansowych:

1. Poważni inwestorzy inwestują regularnie (to dla nich swoisty nawyk).
2. Dywersyfikacja jest kluczem do sukcesu w dłuższym terminie.
3. Najlepsi inwestorzy właściwie równoważą swoje portfele, dzieląc inwestycje na bezpieczne, konserwatywne oraz ryzykowne, ale dające szansę szybkiego pomnożenia kapitału.
4. Inwestorzy starają się kupować tanio i sprzedawać drogo, aby osiągać maksymalny zwrot z inwestycji.

[7] Tzw. **aktywa wygasające** (ang. *expiring assets*) to takie, których wartość maleje w czasie. Innymi przykładami takich aktywów są magazyny pełne bananów czy bilet na mecz koszykówki.

5. Portfele powinny być poddawane analizie i korygowane w regularnych odstępach czasu.

Aby osiągnąć sukces w karierze, musimy zarządzać portfolio swojej wiedzy, postępując według tych samych wskazówek.

Dobra wiadomość jest taka, że zarządzanie tego rodzaju inwestycjami jest taką samą umiejętnością, jak każda inna — można się jej nauczyć. Sztuką jest, aby zacząć to robić, a następnie stworzyć w sobie taki nawyk. Stwórz rutynę i postępuj zgodnie z nią tak długo, aż Twój mózg uzna ją za swoją. Od tej pory będziesz automatycznie „zasysał" nową wiedzę.

Budowa własnego portfolio wiedzy

- **Regularne inwestowanie.** Tak jak w świecie inwestycji finansowych, musimy inwestować w swoje portfolio wiedzy możliwie *regularnie*. Nawet jeśli przedmiotem inwestycji są niewielkie kwoty, odpowiedni nawyk jest równie ważny jak inwestowane sumy. Kilka przykładowych celów zostanie opisanych w następnym podrozdziale.
- **Różnorodność.** Im więcej *różnych* zagadnień znamy, tym większa jest nasza wartość. Absolutnym minimum jest znajomość podstawowych cech technologii, której aktualnie używamy w swojej pracy. Nie możemy jednak na tym poprzestać. Świat komputerów zmienia się na tyle gwałtownie, że technologia bijąca dzisiaj rekordy popularności jutro może być niemal bezużyteczna (a przynajmniej skazana na zapomnienie). Im więcej technologii opanujemy, tym łatwiej będziemy mogli dostosowywać się do zachodzących zmian.
- **Zarządzanie ryzykiem.** Z technologiami wiążą się bardzo różne czynniki ryzyka — istnieją zarówno technologie cechujące się dużym zyskiem przy małym ryzyku, jak i rozwiązania oferujące stosunkowo niewiele w warunkach wysokiego ryzyka. Inwestowanie wszystkich środków w ryzykowne udziały, które mogą z dnia na dzień okazać się bezwartościowe, z pewnością nie byłoby rozsądnym posunięciem. Nie należy też inwestować wszystkich zasobów w najbardziej bezpieczne, zachowawcze rozwiązania, ponieważ można w ten sposób przegapić najlepsze okazje. Nie powinniśmy więc umieszczać wszystkich technicznych jajek w jednym koszyku.
- **Kupuj tanio, sprzedawaj drogo.** Nauka nowych technologii jeszcze przed zyskaniem większej popularności bywa równie trudna jak odnajdywanie niedoszacowanych papierów wartościowych, jednak potencjalna nagroda w obu przypadkach będzie bardzo kusząca. Nauka Javy zaraz po jej pierwszym wydaniu być może była ryzykowna, ale zapewne zwróciła się z nawiązką pionierom tej technologii, którzy mają dzisiaj status najlepiej opłacanych ekspertów.
- **Przeglądy i korekty.** Mamy do czynienia z branżą podlegającą wyjątkowo dynamicznym zmianom. Popularna technologia, której używamy zaledwie od miesiąca, już jutro może okazać się całkowitym przeżytkiem. Być może

warto wrócić do technologii bazy danych, której nie używaliśmy od jakiegoś czasu. Niewykluczone, że uzyskamy nieporównanie lepszą propozycję pracy, jeśli spróbujemy jeszcze opanować język...

Spośród wszystkich tych wskazówek najważniejsza jest ta, która wydaje się najprostsza:

WSKAZÓWKA NR 9

Regularnie inwestuj w swoje portfolio wiedzy.

Cele

Skoro dysponujemy już pewnymi wskazówkami, jak i kiedy uzupełniać nasze portfolio wiedzy, warto zastanowić się nad najlepszymi sposobami pozyskiwania kapitału intelektualnego niezbędnego do wypełnienia tego portfolio. Poniżej przedstawiono kilka sugestii.

- **Warto uczyć się przynajmniej jednego nowego języka rocznie.** Różne języki programowania pozwalają rozwiązywać te same problemy na różne sposoby. Stałe poznawanie nowych rozwiązań ułatwia szersze postrzeganie rozwiązywanych problemów i zmniejsza ryzyko wybierania utartych sposobów postępowania. Co więcej, uczenie się nowych języków jest teraz dużo prostsze dzięki materiałom i oprogramowaniu dostępnym za darmo w internecie.
- **Należy czytać jedną książkę techniczną na miesiąc.** Chociaż w internecie znajduje się mnóstwo krótkich artykułów oraz można w nim znaleźć wartościowe odpowiedzi na nurtujące nas pytania, dokładne zrozumienie tematu wymaga przeczytania książki. Warto poszukać w księgarniach książek technicznych poświęconych zagadnieniom w ten czy inny sposób związanym z aktualnie realizowanym projektem[8]. Kiedy już nabierzemy odpowiednich przyzwyczajeń, powinniśmy sięgać po nową książkę przynajmniej raz w miesiącu. Po opanowaniu aktualnie używanych technologii warto poświęcić trochę czasu na poznawanie rozwiązań *niezwiązanych* z aktualnie realizowanym projektem.
- **Powinniśmy też sięgać po książki inne niż techniczne.** Musimy pamiętać, że komputery są używane przez **ludzi** — ludzi, których potrzeby próbujemy zaspokoić, tworząc odpowiednie oprogramowanie. Nie wolno nam zapominać o stronie równania, po której występuje żywy człowiek.
- **Powinniśmy brać udział w szkoleniach.** Warto sprawdzić, czy lokalne uczelnie nie organizują wartościowych kursów. Niewykluczone, że cenną wiedzę będzie można zdobyć na przykład podczas zbliżających się targów.

[8] Być może to tylko nasze zdanie, ale wartościowa lista znajduje się pod adresem *https://praprog.com*.

- **Należy zaangażować się w funkcjonowanie lokalnych grup użytkowników.** Nie wystarczy tylko iść na spotkanie i słuchać, co inni mają do powiedzenia — chodzi o aktywny udział. Izolacja jest śmiertelnym zagrożeniem dla kariery. Warto więc szukać kontaktów z osobami pracującymi poza naszą firmą.
- **Należy eksperymentować z różnymi środowiskami.** Jeśli pracujemy wyłącznie w systemie Windows, warto spróbować poznać system Unix w domu (wprost doskonałym wyborem będzie któraś z darmowych dystrybucji Linuksa). Jeśli do tej pory korzystaliśmy tylko z plików *makefile* i zwykłego edytora, koniecznie powinniśmy sprawdzić możliwości środowisk IDE (i odwrotnie).
- **Należy trzymać rękę na pulsie.** Warto czytać artykuły i posty online dotyczące technologii innych niż te, których używasz w swoim bieżącym projekcie. W ten sposób można się dowiedzieć, jakie doświadczenia mają inni z tymi technologiami, jakiego używają słownictwa itp.

Niezwykle ważne jest ustawiczne szukanie i pogłębianie wiedzy. Kiedy uznajemy, że opanowaliśmy nowy język lub technologię w dostatecznym stopniu, powinniśmy iść dalej. To doskonały moment, by nauczyć się czegoś nowego.

Nie ma znaczenia, czy kiedykolwiek używaliśmy którejś z tych technologii w ramach jakiegoś projektu ani nawet czy wspominaliśmy o tej technologii w swoich CV. Proces uczenia się poszerza nasze horyzonty myślowe, stwarzając nowe możliwości i wskazując nowe drogi osiągania celów. W tym fachu bardzo ważne jest umiejętne łączenie wiedzy z różnych źródeł. Warto więc próbować wykorzystywać nowe umiejętności już w trakcie bieżącego projektu. Nawet jeśli ten projekt jest realizowany w innej technologii, być może istnieje możliwość zastosowania przynajmniej niektórych pomysłów. Wystarczy opanować na przykład programowanie obiektowe, aby nieco zmienić styl programowania. Zrozumienie paradygmatu programowania funkcyjnego pozwoli pisać kod obiektowy inaczej.

Okazje do nauki

Przyjmijmy, że łapczywie sięgamy po wszystkie dostępne materiały i doskonale orientujemy się w nowinkach w naszej dziedzinie (co nie jest proste). Załóżmy, że nagle ktoś zadaje nam jakieś pytanie. Nie mamy zielonego pojęcia, jak na nie odpowiedzieć, do czego od razu przyznajemy się swojemu rozmówcy.

Nie możemy na tym poprzestać. Powinniśmy raczej wykorzystać sytuację i traktować znalezienie odpowiedzi jako swoiste wyzwanie. Możemy zapytać innych, poszukać w internecie — warto czytać także artykuły naukowe, nie tylko opinie użytkowników.

Jeśli sami nie możemy znaleźć odpowiedzi, powinniśmy poszukać osoby, która *poradzi sobie* z tym zadaniem. Nie wolno nam tak tego zostawić. Rozmowy z innymi ułatwią nam budowę sieci osobistych relacji. W ten sposób nierzadko

można odnajdywać — ku własnemu zdziwieniu — rozwiązania innych problemów (niezwiązanych z początkowym tematem rozmów). Wszystko to sprawia, że nasze portfolio stale jest powiększane…

Lektura wszystkich tych materiałów i poszukiwanie wiedzy z natury rzeczy wymaga czasu. Właśnie czas jest tutaj zasobem deficytowym. Oznacza to, że musimy planować swoje poczynania z wyprzedzeniem. Warto zawsze mieć pod ręką coś do przeczytania na wypadek, aby nigdy nie siedzieć bezczynnie. Czas spędzany w poczekalniach u lekarzy lub dentystów to wprost doskonała okazja do nadrobienia zaległości w czytaniu. Nie możemy jednak liczyć na innych — jeśli nie mamy ze sobą interesujących nas materiałów, możemy skończyć, trzymając w dłoniach pomięty miesięcznik z 1973 roku pełen artykułów o Papui-Nowej Gwinei.

Krytyczne myślenie

Ostatnim ważnym punktem jest *krytyczna* ocena tego, co czytamy i słyszymy. Musimy mieć pewność, że wiedza, która trafia do naszego portfolio, jest prawidłowa i nie została zniekształcona przez marketingowy przekaz producenta czy mediów. Należy wystrzegać się fanatyków dogmatycznie przywiązanych do *swoich* racji — ich poglądy mogą, ale nie muszą potwierdzić się w naszym przypadku (i w ramach realizowanego projektu).

Nigdy nie powinniśmy lekceważyć siły komercjalizacji. To, że wyszukiwarka internetowa wyświetla coś na pierwszym miejscu, nie oznacza jeszcze, że trafiliśmy na najlepszą stronę; być może jej właściciel po prostu zapłacił za wyższą pozycję w wynikach. To, że księgarnia wystawia jakąś książkę na witrynie, nie oznacza jeszcze, że książka jest dobra (ani nawet popularna); być może jej wydawca zapłacił za eksponowanie swojego tytułu.

WSKAZÓWKA NR 10

Patrz krytycznym okiem na to, co czytasz i słyszysz.

Krytyczne myślenie to oddzielna dyscyplina. Zachęcamy do przeczytania i przestudiowania wszystkiego, co uda Ci się zdobyć na ten temat. Zanim rozpoczniesz lekturę, oto lista z kilkoma pytaniami, które warto sobie zadać i przeanalizować odpowiedzi.

- **Zadawaj pytania: „pięć razy dlaczego”.** Ulubiona sztuczka konsultingowa: zadaj pytanie „dlaczego?” co najmniej pięć razy. Zadaj pytanie i uzyskaj odpowiedź. Przeanalizuj problem dokładniej, dalej pytając „dlaczego?”. Powtarzaj to pytanie tak, jakbyś był czterolatkiem (ale grzecznym). Być może w ten sposób uda Ci się zbliżyć do poznania głównej przyczyny problemu.

- **Kto na tym skorzysta?** Choć może się to wydawać cyniczne, *podążanie za pieniędzmi* może być bardzo pomocną ścieżką do analizy. Korzyści kogoś innego lub innej organizacji mogą być tożsame z Twoimi własnymi — bądź nie.
- **Jaki jest kontekst?** Weźmy za przykład artykuł lub książkę reklamującą „najlepsze praktyki". Dobre pytanie do rozważenia brzmi: „najlepsze dla kogo?". Jakie są warunki wstępne, jakie są konsekwencje, zarówno krótko-, jak i długoterminowe?
- **Kiedy i w jakich warunkach rozwiązanie się sprawdza?** Zastanów się nad okolicznościami. Czy jest za późno? A może za wcześnie? Nie poprzestawaj na myśleniu wyłącznie o następnym kroku (**co wydarzy się za chwilę**), ale myśl także długoterminowo (**co zdarzy się w dalszej kolejności**).
- **Dlaczego to jest problem?** Czy istnieje model bazowy? W jaki sposób ten model działa?

Praktyka pokazuje, że prostych odpowiedzi jest bardzo niewiele. Odpowiednio bogate portfolio i krytyczne spojrzenie na otaczającą nas masę publikacji powinny jednak umożliwić nam wypracowywanie odpowiedzi nawet na najtrudniejsze pytania.

Pokrewne podrozdziały

- Temat 1., „To jest Twoje życie".
- Temat 22., „Dzienniki inżynierskie".

Wyzwania

- Naukę nowego języka powinniśmy rozpocząć już w tym tygodniu. Od zawsze programujesz w tym samym, starym języku? Spróbuj więc napisać coś w językach Clojure, Elixir, Elm, F#, Go, Haskell, Python, R, ReasonML, Ruby, Rust, Scala, Swift, TypeScript lub dowolnym innym, który zwrócił Twoją uwagę i (lub), który Ci się spodobał[9].
- Sięgnijmy po nową książkę (ale dopiero po skończeniu tego wydania). Jeśli aktualnie zajmujemy się szczegółową implementacją i kodowaniem, powinniśmy przeczytać książkę o projektowaniu i architekturze. Jeśli natomiast pracujemy nad wysokopoziomowym projektem, dla odmiany powinniśmy sięgnąć po książkę o technikach kodowania.

[9] Nigdy nie słyszałeś o żadnym z tych języków? Pamiętaj, że wiedza, podobnie jak popularna technologia, jest wygasającym atutem. Lista nowych i eksperymentalnych języków była zupełnie inna w pierwszym wydaniu i prawdopodobnie będzie inna w chwili, gdy będziesz czytać tę książkę. Tym bardziej masz powód, aby nie przestawać się uczyć.

- Powinniśmy wyjść i porozmawiać o technologiach z osobami, które nie są zaangażowane w bieżący projekt lub które w ogóle nie pracują w naszej firmie. Warto zawierać nowe znajomości w firmowej kafejce lub poszukać entuzjastów podobnych do nas na lokalnym spotkaniu grupy użytkowników.

7 Komunikuj się!

Wierzę, że lepiej być przedmiotem krytycznego osądu niż niezauważonym.

Mae West, *Piękność lat dziewięćdziesiątych*, 1934

Być może możemy się czegoś nauczyć od pani West. Nie chodzi tylko o to, co mamy, ale też o to, jak to zapakujemy. Nawet najlepsze pomysły, najdoskonalszy kod i najbardziej pragmatyczne myślenie będą jałowe, jeśli nie nauczymy się komunikacji z innymi ludźmi. Bez efektywnej komunikacji nawet dobra idea staje się sierotą.

Jako programiści musimy komunikować się na wielu poziomach. Spędzamy całe godziny na spotkaniach, podczas których słuchamy i mówimy. Pracujemy z użytkownikami końcowymi, próbując zrozumieć ich potrzeby. Piszemy kod, którego zadaniem jest zarówno komunikowanie naszych intencji maszynie (komputerowi), jak i dokumentowanie naszych przemyśleń przyszłym pokoleniom programistów. Piszemy propozycje i notatki dołączane do wniosków o zasoby i wyjaśniających ich stosowanie, raportujące postępy prac oraz sugerujące nowe kierunki. Co więcej, codziennie pracujemy z naszymi zespołami, próbując przekonywać je do naszych pomysłów, zmieniać dotychczasowe praktyki ich członków i sugerować nowe rozwiązania. Ponieważ znaczną część czasu pracy poświęcamy właśnie na komunikację, musimy robić to naprawdę dobrze.

Traktuj angielski (lub dowolny język naturalny, którym się posługujesz) jak specjalny język programowania. Pisz w języku naturalnym tak, jak pisze się kod, stosuj zasady DRY, ETC, narzędzia automatyzacji itp. (zasady projektowania DRY i ETC omówimy w następnym rozdziale).

WSKAZÓWKA NR 11

Język naturalny to po prostu kolejny język programowania.

W tym podrozdziale sporządzimy listę sugestii, które mogą być pomocne podczas doskonalenia umiejętności komunikacyjnych.

Poznaj swoich odbiorców

Komunikujesz się tylko wtedy, gdy przekazujesz to, co chcesz przekazać, samo mówienie nie wystarczy. Aby to robić, musisz rozumieć potrzeby, zainteresowania i możliwości swoich odbiorców. Wszyscy uczestniczyliśmy w spotkaniach, w czasie których jakiś maniakalny programista z błyskiem w oczach wygłaszał wiceprezesowi ds. marketingu długi monolog na temat meritum jakiejś tajemnej technologii. To nie jest komunikacja: to tylko monolog, i do tego denerwujący[10].

Załóżmy, że chcesz zmienić zdalny system monitorowania, aby używać zewnętrznego brokera komunikatów do publikowania powiadomień o stanie. W zależności od odbiorców informacje można przedstawić na wiele różnych sposobów. Użytkownicy końcowi docenią, że ich systemy będą teraz współdziałać z innymi usługami korzystającymi z brokera. Twój dział marketingu będzie mógł wykorzystać ten fakt, aby zwiększyć sprzedaż. Menedżerowie działów programowania i operacyjnego będą zadowoleni, ponieważ utrzymanie tej części systemu będzie odtąd problemem kogoś innego. Wreszcie programiści mogą korzystać z doświadczeń związanych z posługiwaniem się nowymi interfejsami API, a może nawet będą w stanie znaleźć nowe zastosowania dla brokera komunikatów. Dzięki odpowiedniemu sposobowi dotarcia do każdej grupy, członkowie każdej z nich będą podekscytowani Twoim projektem.

Podobnie jak w przypadku wszystkich form komunikacji, sztuką jest zebranie opinii. Nie czekaj biernie na pytania: poproś o ich stawianie. Zwracaj uwagę na język ciała i mimikę odbiorców. Jednym z założeń programowania neurolingwistycznego jest zasada „Sensem Twojej komunikacji jest odpowiedź, którą otrzymujesz". Gdy się komunikujesz, stale poszerzaj wiedzę o swoich odbiorcach.

Należy wiedzieć, co powiedzieć

Bodaj najtrudniejszym aspektem oficjalnych form komunikacji obowiązujących w biznesie jest precyzyjne określanie, co naprawdę chcemy powiedzieć. O ile autorzy beletrystyki mogą pozwolić sobie na szczegółowe planowanie swoich dzieł przed przystąpieniem do ich tworzenia, ludzie odpowiedzialni za tworzenie dokumentacji technicznej nierzadko muszą w jednej chwili usiąść przed komputerem i niezwłocznie zacząć spisywać (począwszy od „Rozdział 1. Wprowadzenie") wszystko, co przyjdzie im do głowy.

Musimy zaplanować, co chcemy powiedzieć. Powinniśmy przygotować szkic przyszłej wypowiedzi. Warto też odpowiedzieć sobie na pytanie, czy planowana wypowiedź rzeczywiście wyraża to, co chcemy przekazać rozmówcom. Należy doskonalić plan wypowiedzi tak długo, aż odpowiedź na to pytanie będzie satysfakcjonująca.

[10] Angielskie słowo *annoy* (denerwujący) pochodzi od starofrancuskiego *enui*, które oznacza również „przynudzać".

Proponowane rozwiązania nie sprawdzają się podczas pisania dokumentów. Przed pójściem na ważne spotkanie lub odbyciem rozmowy telefonicznej z ważnym klientem warto zanotować sobie pomysły, które mamy zakomunikować, i zaplanować kilka strategii ich prezentacji.

Teraz, gdy już wiesz, czego chcą Twoi odbiorcy, spróbuj spełnić ich oczekiwania.

Należy wybrać właściwy moment

Jest piątek, godzina 18. Wszyscy uczestnicy spotkania mają za sobą trudny tydzień. Najmłodsze dziecko szefa jest w szpitalu, na zewnątrz leje jak z cebra, a powrót do domu w piątkowy wieczór będzie prawdziwym koszmarem. Prawdopodobnie nie jest to najlepszy moment na rozmowę z szefem o konieczności rozbudowy komputera.

Aby dobrze zrozumieć, co powinni usłyszeć nasi rozmówcy, musimy zidentyfikować ich priorytety. Na przykład pomysły dotyczące repozytoriów kodu źródłowego najlepiej zaprezentować menedżerowi, który właśnie odbył przykrą rozmowę z własnym przełożonym po utracie części kodu źródłowego — mamy wówczas spore szanse, że nasze propozycje trafią na podatny grunt. Zarówno treść naszych propozycji, jak i moment ich prezentacji muszą odpowiadać na bieżące problemy firmy. Nierzadko wystarczy po prostu zadać sobie pytanie: Czy to dobry moment, aby o tym rozmawiać?

Należy wybrać odpowiedni styl

Styl przekazu należy dostosować do charakteru jego odbiorców. Część ludzi oczekuje formalnych prezentacji ograniczających się wyłącznie do faktów. Inni wolą poprzedzać właściwe rozmowy biznesowe długimi pogawędkami na najróżniejsze tematy. Podobnie jest w przypadku dokumentów pisanych — niektórzy wolą długie raporty z szerokimi wyjaśnieniami, inni oczekują raczej krótkich notatek lub zwięzłych wiadomości poczty elektronicznej. W razie wątpliwości warto zapytać.

Należy jednak pamiętać, że sami stanowimy połowę tej swoistej transakcji komunikacyjnej. Jeśli ktoś oczekuje akapitu opisującego jakiś aspekt i jeśli wiemy, że do opisania tego złożonego aspektu będziemy potrzebowali kilku stron, należy o tym po prostu powiedzieć. Musimy pamiętać, że także reakcje na propozycje same w sobie stanowią formę komunikacji.

Należy zadbać o warstwę estetyczną

Nasze pomysły są ważne. Zasługują więc na odpowiednią oprawę, aby lepiej trafić do odbiorców.

Zbyt wielu programistów (wraz ze swoimi menedżerami) koncentruje się wyłącznie na treści tworzonych przez siebie dokumentów. Uważamy to za błąd. Każdy dobry kucharz (lub widz programów kulinarnych) wie, że można zamknąć się w kuchni na wiele godzin, by następnie w jednej chwili zepsuć cały efekt wskutek kiepskiej prezentacji.

W dzisiejszych czasach nic nie może usprawiedliwić kiepskiego wyglądu drukowanych dokumentów. Współczesne edytory tekstu umożliwiają tworzenie wprost doskonałych dokumentów niezależnie od tego, czy piszesz używając notacji Markdown, czy stosujesz procesor tekstu. Wystarczy opanować zaledwie kilka prostych poleceń. Jeśli używany przez nas edytor obsługuje arkusze stylów, koniecznie powinniśmy skorzystać z tej możliwości. (Być może nasza firma zdefiniowała już arkusze stylów, których możemy używać w swoich dokumentach). Warto nauczyć się ustawiania nagłówków i stopek stron. Jeśli brakuje nam pomysłów i koncepcji układu dokumentu, wystarczy zajrzeć do przykładowych dokumentów dołączonych do danego edytora. Należy jeszcze *sprawdzić pisownię* (najpierw przy użyciu automatycznego narzędzia, a następnie ręcznie). Istnieją kończ obłędy, których nie morze wychwyci żadne automatyczny mechanizmu.

Należy zaangażować odbiorców

Nierzadko odkrywamy, że proces pracy nad dokumentem jest pod wieloma względami cenniejszy niż sam dokument. Jeśli to możliwe, warto zaangażować przyszłych czytelników w prace już nad wczesnymi wersjami dokumentu. Warto zebrać ich opinie i skorzystać z ich rad. Opisany tryb przygotowywania materiałów pozwoli nie tylko zbudować lepsze relacje ze współpracownikami, ale też tworzyć lepsze dokumenty.

Należy słuchać innych

Istnieje prosta technika, którą musimy stosować, jeśli chcemy być słuchani przez innych — *musimy sami ich słuchać*. Nawet w sytuacji, w której dysponujemy wszystkimi informacjami, nawet jeśli uczestniczymy w formalnym spotkaniu z dwudziestoma wysoko postawionymi menedżerami — jeśli nie słuchamy innych, oni nie będą słuchali nas.

Warto zachęcać ludzi do mówienia, zadając im pytania lub prosząc o streszczenie tego, o czym sami mówiliśmy. Przekształcenie spotkania w dialog znacznie podniesie efektywność naszego przekazu. Kto wie, być może nawet czegoś się nauczymy.

Należy wracać do rozmówców

Kiedy zadajemy komuś pytanie, brak jakiejkolwiek odpowiedzi traktujemy jako przejaw złego wychowania. Czy jednak sami nie ignorujemy wiadomości poczty elektronicznej lub notatek otrzymywanych od osób proszących o jakieś infor-

macje lub oczekujących jakichś czynności z naszej strony? W dzisiejszym świecie łatwo o tym zapomnieć. Zawsze powinniśmy odpowiadać na wiadomości poczty elektronicznej i nagrania na automatycznej sekretarce, nawet jeśli ta odpowiedź będzie brzmiała „wrócę do tego później". Informowanie innych o zainteresowaniu podnoszonymi problemami przekłada się na dużo większą wyrozumiałość w przypadku sporadycznych błędów i upewnia współpracowników w przekonaniu o tym, że ktoś pamięta o ich sprawach.

WSKAZÓWKA NR 12

Ważne jest nie tylko to, co mówimy, ale też to, jak to mówimy.

Jeśli nie pracujemy w próżni, musimy opanować sztukę komunikacji. Im bardziej efektywna będzie nasza komunikacja ze współpracownikami, tym większy będzie nasz wpływ na otaczającą nas rzeczywistość.

Dokumentacja

Wreszcie pozostaje kwestia komunikowania się za pośrednictwem dokumentacji. Zazwyczaj programiści nie poświęcają dokumentacji zbyt wiele czasu. W najlepszym razie jest to nieprzyjemna konieczność; w najgorszym przypadku jest traktowana jako zadanie o niskim priorytecie, a opracowujący dokumentację mają nadzieję, że kierownictwo o niej zapomni pod koniec projektu.

Pragmatyczni programiści uwzględniają dokumentację jako integralną część całego cyklu życia projektu. Jej pisanie można ułatwić. Nie trzeba powielać pracy lub marnować czasu. Dokumentacja może być cały czas pod ręką — w kodzie źródłowym.

WSKAZÓWKA NR 13

Buduj dokumentację wraz z projektem zamiast ją do niego „przytwierdzać".

Dobrze wyglądającą dokumentację można łatwo stworzyć na podstawie komentarzy w kodzie źródłowym. Zachęcamy do dodawania komentarzy do modułów i eksportowanych funkcji, aby ułatwić innym programistom korzystanie z naszego kodu.

Nie oznacza to jednak, że zgadzamy się z osobami, które mówią, że każda funkcja, struktura danych, deklaracja typu itp., potrzebuje osobnego komentarza. Pisanie tego rodzaju mechanicznych komentarzy faktycznie utrudnia utrzymanie kodu: po wprowadzeniu zmian są dwie rzeczy do aktualizacji — zarówno kod, jak i komentarze. Zatem ogranicz komentarze niezwiązane z API do wyjaśniania **dlaczego** coś zostało zrobione — jaki jest tego cel i przeznaczenie. To kod powinien pokazywać **jak** zadanie zostało wykonane, więc zbyt szczegółowe jego komentowanie jest zbędne i stanowi naruszenie zasady DRY.

Komentowanie kodu źródłowego daje doskonałą okazję do udokumentowania tych nieuchwytnych fragmentów projektu, których nie można udokumentować nigdzie indziej: kompromisów inżynierskich, powodów podejmowania decyzji, odrzuconych alternatyw itp.

Podsumowanie

- Należy wiedzieć, co powiedzieć.
- Należy wiedzieć coś o rozmówcach.
- Należy wybrać właściwy moment.
- Należy wybrać odpowiedni styl.
- Należy zadbać o warstwę estetyczną.
- Należy zaangażować odbiorców.
- Należy słuchać innych.
- Należy wracać do rozmówców.
- Należy utrzymywać kod razem z dokumentacją.

Pokrewne podrozdziały

- Temat 15., „Szacowanie”.
- Temat 18., „Efektywna edycja”.
- Temat 45., „Kopalnia wymagań”.
- Temat 49., „Pragmatyczne zespoły”.

Wyzwania

- Istnieje wiele dobrych książek, których fragmenty poświęcono komunikacji w ramach zespołów projektowych. Należą do nich *The Mythical Man-Month: Essays on Software Engineering* [Bro96] oraz *Peopleware: Productive Projects and Teams* [DL13]. Warto przeczytać te pozycje w ciągu najbliższych 18 miesięcy. W książce *Trudni współpracownicy* [BR89] dodatkowo omówiono problem bagażu emocjonalnego, który każdy z nas wnosi do swojego środowiska pracy.
- Kiedy przy najbliższej okazji będziemy przygotowywali prezentację lub pisali notatkę przekonującą odbiorcę do jakiegoś stanowiska, powinniśmy raz jeszcze przestudiować przedstawione w tym rozdziale rady. Należy wyraźnie zidentyfikować odbiorców oraz to, co chcemy im przekazać. Jeśli to możliwe, warto porozmawiać z odbiorcami po prezentacji, aby dowiedzieć się, na ile słuszne były założenia dotyczące ich potrzeb i oczekiwań.

Komunikacja online

Wszystko, co do tej pory napisano o komunikacji przy użyciu dokumentów, ma zastosowanie także w przypadku poczty elektronicznej, postów w mediach społecznościowych, blogach itp. W szczególności poczta elektroniczna osiągnęła status podstawowej platformy komunikacji w ramach korporacji i pomiędzy korporacjami. Poczta elektroniczna służy dzisiaj do negocjowania kontraktów i prowadzenia sporów, a nierzadko stanowi ważny dowód w sądzie. Okazuje się jednak, że z jakiegoś powodu ludzie, którzy nigdy nie wysłaliby papierowego dokumentu z najmniejszym niedociągnięciem, beztrosko wysyłają w świat niechlujne, wręcz odpychające wiadomości poczty elektronicznej.

Wskazówki dotyczące poczty elektronicznej są dość proste:

- Należy przeczytać tekst przed kliknięciem przycisku *Wyślij*.
- Należy sprawdzić pisownię.
- Należy zachować prosty format.
- Należy ograniczać liczbę cytatów. Nikt nie lubi otrzymywać w całości własnej stuwierszowej wiadomości z lakoniczną odpowiedzią „Masz rację"!
- Jeśli już cytujemy cudzą wiadomość poczty elektronicznej, koniecznie powinniśmy odpowiednio wyróżnić cytat i umieścić go w tekście (nie w załączniku). To samo dotyczy cytowania w mediach społecznościowych.
- Nie należy przeklinać; w przeciwnym razie nasze przekleństwa będą nas jeszcze długo prześladowały.
- Przed wysłaniem wiadomości warto jeszcze raz sprawdzić listę adresatów. Byłoby niezręcznie, gdybyś skrytykował swojego przełożonego w wiadomości rozesłanej do pracowników własnego działu, zapominając, że na liście odbiorców jest także krytykowany szef. Jeszcze lepiej będzie, jeśli w ogóle nie będziemy krytykować szefa w wysyłanym e-mailu.

Jak przekonało się wielu pracowników licznych korporacji oraz polityków, wiadomości poczty elektronicznej i posty w mediach społecznościowych są wieczne. Należy przykładać do nich taką samą wagę jak do tradycyjnych notatek i raportów.


PROGRAM
PARTNERSKI
GRUPY HELION
1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ
Zmień swoją stronę WWW w działający bankomat!
Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl
GRUPA Helion

Dbaj o swoje rzemiosło i myśl o tym, co robisz!

Programiści dysponują coraz lepszym, szybszym i wszechstronniejszym sprzętem. Pojawiają się nowe języki programowania i nowe paradygmaty tworzenia architektury oprogramowania. Są jednak rzeczy, które w świecie programowania pozostają stałe i niezmienne. Wciąż proces stawania się programistą wymaga od adeptów tego rzemiosła sporego wysiłku. Akt kodowania to za mało. Trzeba zmienić sposób myślenia, nawyki, zachowania i oczekiwania. Konieczne jest świadome dążenie do stosowania dobrych praktyk. Jeśli pilnuje się jakości swojej pracy i nieustannie pamięta, co i po co się robi, można w końcu stać się pragmatycznym programistą.

W drugim wydaniu tego kultowego przewodnika wskazówki techniczne harmonijnie łączą się z aspektami filozofii pragmatycznego programisty. Książka została zaktualizowana i gruntownie przejrzana, aby teraz, dwadzieścia lat po pierwszym wydaniu, ponownie pokazać, co to znaczy być nowoczesnym, pragmatycznym programistą. Poruszono tu tematy osobistej odpowiedzialności i rozwoju zawodowego, komunikacji i poznawania prawdziwych wymagań, nowoczesnych technik architektonicznych oraz coraz ważniejszych kwestii zachowania bezpieczeństwa i prywatności. Książka składa się z krótkich rozdziałów, które tworzą szeroki kontekst, dzięki czemu zyskasz wiedzę o najlepszych podejściach, unikniesz głównych pułapek, a co najważniejsze — rozwiniesz nawyki i postawy, które staną się fundamentem Twojego sukcesu zawodowego.

Dowiedz się, jak:

- pisać kod dynamiczny, elastyczny i łatwy do dostosowywania
- unikać pułapek związanych z powielaniem wiedzy
- chronić oprogramowanie przed lukami w zabezpieczeniach
- budować zespoły pragmatycznych programistów
- skutecznie testować
- wziąć odpowiedzialność za swoją pracę i karierę

David Thomas jest programistą, autorem książek i redaktorem. Występował na wielu prestiżowych konferencjach programistycznych. Mieszka niedaleko Dallas w Teksasie.
Andrew Hunt jest autorem znakomitych książek o tworzeniu oprogramowania.
Jego pasją jest gra na trąbce i instrumentach klawiszowych.

Obaj autorzy uczestniczyli w tworzeniu oryginalnego *Manifestu Agile* i założeniu *Agile Alliance*. Współtworzyli też serię książek *The Pragmatic Bookshelf*.

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
HELIONSZKOLENIA.PL
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-283-7139-2
9 788328 371392
INFORMATYKA W NAJLEPSZYM WYDANIU
Cena: 77,00 zł
Pearson
Addison-Wesley