# Sprzedaj swój program. Droga do udanych projektów programistycznych


Autorzy: Jared R. Richardson, William A. Gwaltney Jr.
Tłumaczenie: Dariusz Biskup
ISBN: 83-246-0408-1
Tytuł oryginału: Ship it! A Practical Guide to Successful Software Projects
Format: B5, stron: 256

**Stwórz niezawodne oprogramowanie spełniające oczekiwania użytkowników**

- Wykorzystuj odpowiednie narzędzia projektowe.
- Wdrażaj nowoczesne metodologie.
- Szybko rozwiązuj problemy.

Dyskusje nad wadami i zaletami przeróżnych metodologii tworzenia oprogramowania, mające na celu wyłonienie najlepszej z nich, zwykle do niczego nie prowadzą. Zwolennicy poszczególnych metodologii, takich jak Rational Unified Process, programowanie ekstremalne i inne, starają się udowodnić, że to ich stanowisko jest poprawnym sposobem realizacji projektów informatycznych. Tymczasem nie istnieje „jedyne słuszne" i uniwersalne podejście, które sprawdza się we wszystkich okolicznościach. Wybór właściwej metodologii w ogromnej mierze zależy od typu projektu i wielkości zespołu pracującego nad nim. Należy kierować się nastawieniem czysto pragmatycznym, czyli wybrać taką metodologię, która będzie najbardziej korzystna dla określonego projektu. Niewłaściwy wybór może skończyć się porażką.

Książka „Sprzedaj swój program. Droga do udanych projektów programistycznych" to zbiór wskazówek przedstawiających narzędzia i techniki, dzięki którym każdy projekt programistyczny zakończy się sukcesem. Czytając ją, nauczysz się korzystać z nowoczesnych instrumentów wykorzystywanych do projektowania oprogramowania, kontroli wersji kodu źródłowego i śledzenia procesu usuwania błędów. Dowiesz się, w jaki sposób zorganizować pracę zespołu projektowego i wdrażać metodologię wytwarzania oprogramowania. Porady, które znajdziesz w tej książce, pomogą Ci rozwiązać problemy pojawiające się podczas realizacji projektów programistycznych. Poznasz nowoczesne metody oraz dowiesz się, kiedy i jak z nich korzystać.

- Planowanie infrastruktury
- Dobór narzędzi projektowych
- Automatyzacja zadań
- Tworzenie listy zadań
- Rola kierownika technicznego
- Metodologia pocisku smugowego
- Rozwiązywanie problemów

Wskazówki zawarte w tej książce sprawią, że każdy prowadzony przez Ciebie projekt zakończy się w terminie i zmieści w wyznaczonym budżecie.

Wydawnictwo Helion
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

# Spis treści

*Nigdy nie myl działania z postępem.*

Autor nieznany

# Rozdział 3.
# Pragmatyczne techniki projektowe

Dlaczego niektórzy produkują oprogramowanie wysokiej jakości w terminie lub przed terminem, podczas gdy inni (wielu innych, niestety) spóźniają się, przekraczają budżet lub zupełnie rezygnują? Istnieje wiele teorii tłumaczących ten stan rzeczy i wiele metodologii próbujących go poprawić. Jednym z fundamentów tych metod jest poprawa *współpracy*.

*współpraca*

Jeśli nie piszesz swoich programów w garażu, będziesz współpracował z innymi. A nawet *jeśli* piszesz swój program w domu, płaci Ci za niego Twój klient. Większość z nas pracuje w ramach zespołu programistów. Pracujemy z naszymi kolegami, wspólnie pisząc programy.

Chociaż dobra współpraca nie gwarantuje powodzenia w projekcie, zła współpraca niemal na pewno oznacza jego niepowodzenie.

W niniejszym rozdziale opiszemy kilka technik pracy grupowej ułatwiających sprawne i efektywne zarządzanie projektem. Dzięki tym technikom staniesz się bardziej świadomy stanu swojego projektu, jak również postępów osiąganych przez Twoich współpracowników. Spowodują one, że Twoja praca stanie się łatwiejsza, efektywniejsza i będzie Ci sprawiać większą satysfakcję.

Nie istnieje jednolita lista technik poprawiających funkcjonowanie projektu. Każdy projekt i każdy zespół pracowników jest inny. Sztuką jest usprawnianie zarówno procesu tworzenia oprogramowania, jak i zrozumienie efektywnej pracy grupowej.

Rozwijając swoje umiejętności, zawsze natkniesz się na jakąś sztuczkę, której nie znałeś, oraz jakiś nawyk, którego powinieneś się pozbyć. Techniki, które tutaj opisujemy, uważamy za najbardziej przydatne. Obejmują one między innymi:

- listę zadań,
- kierownika technicznego,
- codzienne spotkania,
- przeglądy kodu,
- powiadomienia o zmianie kodu.

Być może pamiętacie te wytyczne jako część większego rysunku na stronie 20, który tutaj pokazany jest we fragmencie (patrz rysunek 3.1).

## 10. Praca przy użyciu listy zadań

Często do koordynowania własnej pracy używamy list zadań. Lista taka może być sformalizowana tak, aby mogła być wykorzystywana w pracy grupowej.

Dawniej, pracując nad mniejszymi projektami, aby koordynować naszą pracę, korzystaliśmy często ze zwykłych notesów. Lista zadań wywodzi się z naszych osobistych notatek o rzeczach, których nie chcieliśmy zapomnieć.


**Rysunek 3.1** Techniki projektowe

Gdy przechodziliśmy na funkcje kierownicze, lista zadań zaczęła zawierać pozycje dotyczące całego zespołu programistów i wówczas notatki na papierze stały się nieefektywne. Po wielokrotnym pokazywaniu listy zadań współpracownikom zaczyna się poszukiwać alternatywnych sposobów jej zapisywania i udostępniania. Tablica funkcjonuje dobrze dla małych zespołów, zwłaszcza w dużym pomieszczeniu. Obecnie staramy się korzystać ze stron internetowych lub wiki. Niektórzy do tworzenia list zadań wykorzystują arkusze kalkulacyjne[1]. Każdy ma dostęp do stron internetowych, a ich edycja jest bardzo łatwa.

Za pomocą listy zadań ustalasz swój dzienny i tygodniowy plan zajęć. Za jej pomocą porządkujesz pracę swoją i całego zespołu. Jeśli jesteś przytłoczony pracą, nie wiesz, za co się zabrać, przeglądasz swoją listę zadań, aby znaleźć punkt zaczepienia. Jeśli utknąłeś przy pracy nad jakimś trudnym problemem i chcesz na chwilę zająć się czymś innym, lista zadań wskazuje Ci pozycje, których możesz użyć jako alternatywnego zajęcia. W ten sposób będziesz zawsze pracował nad najważniejszymi problemami.

[1] Joel Spolsky korzysta z arkusza Excela do edycji swojej listy zadań. Przedstawia również wiele argumentów za tym rozwiązaniem. Więcej informacji na stronie *http://www.joelonsoftware.com/articles/fog0000000245.html*.

## Do czego potrzebna jest lista zadań?

Jak często pracowałeś nad projektem, w którym każdy był zawsze czymś zajęty, a który nigdy się nie kończył? Ważne funkcje były zapominane, pracownicy bezczynnie czekali na te, które nie były gotowe, programiści nie wiedzieli, czym się zająć w dalszej kolejności.

Ktoś musi zapisać wszystkie funkcje, nadać im priorytet i zlecać pracownikom do wykonania zadania znajdujące się na pierwszej pozycji listy zadań. Tworzony jest centralny punkt organizacji pracy Twojego zespołu pozbawiony niedogodności związanych z korzystaniem z bardziej złożonych procedur.

Pracownicy, którzy korzystają z listy zadań, nigdy nie są bezczynni. Kiedy zakończą swoje aktualne zadanie, sprawdzają listę, wybierają zadanie, któremu przypisano wysoki priorytet, i biorą się do pracy. Programiści mogą wybrać swoje następne zadanie spośród tych, które są dla nich najbardziej interesujące, ale kierownik techniczny powinien mieć pewność, że te o najwyższym priorytecie są w toku lub na najwyższej pozycji na liście.

Ponieważ programiści mają różne umiejętności, kierownik techniczny może zgadzać się na wyjątki, ale zasadniczo pozycje drugorzędne nie powinny być wykonywane, dopóki nie zostaną zakończone zadania o najwyższym priorytecie.

Lista zadań (jako narządzie pracy grupowej) daje kierownictwu oraz klientom możliwość oceny produktu przed poniesieniem nakładów na nowe funkcje. Zawsze taniej jest usunąć nową funkcję, zanim spędzi się tydzień nad jej programowaniem! Ile razy zdarzyło Ci się skończyć program i usłyszeć od klienta: „Program jest w porządku, ale byłoby świetnie, jeśli zamiast funkcji X, Y i Z zostałaby dodana funkcja A”? Przy wykorzystaniu listy zadań tworzony jest zarys dokumentacji, która może być udostępniana innym we wczesnych fazach rozwoju produktu.

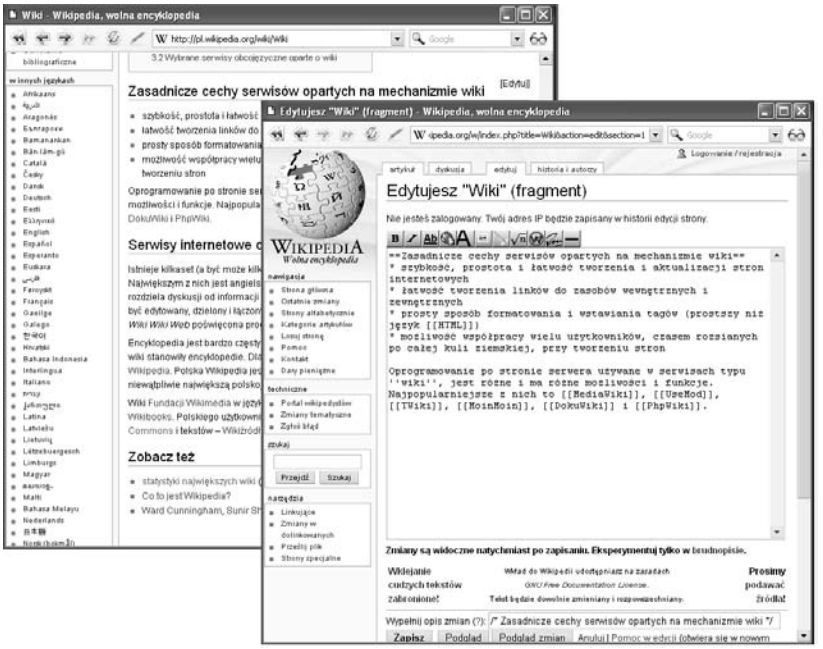
Lista zadań wprowadza również w zespole dużą elastyczność. Zapewnia, że najpierw wykonane będą pewne prace projektowe, gdyż wymaga podzielenia programu na funkcje, a funkcje na pozycje listy. Ponadto, ponieważ program jest podzielony na funkcje, możliwe jest w razie potrzeby usuwanie lub dodawanie nowych pozycji.

## Jaś pyta...

### Co to jest wiki?

*Wiki* jest to prosta metoda tworzenia stron internetowych. Strony internetowe są pisane w uproszczony sposób tak, aby każdy mógł modyfikować ich zawartość. Strony wiki zaprojektowane są w sposób zachęcający do szerokiej współpracy.

Strona typu wiki wygląda na początku dokładnie jak każda inna. Różnica polega na tym, że istnieje na niej odsyłacz „Edycja strony". Kliknięcie go powoduje wyświetlenie prostego edytora tekstowego pozwalającego na bardzo łatwe modyfikowanie strony.




Więcej informacji na stronie *http://pl.wikipedia.org/*.

W każdej firmie znajdzie się osoba (lub osoby), która ma w zwyczaju wpadać niespodziewanie i pytać, dlaczego pewna funkcja nie jest jeszcze zakończona albo dlaczego nikt nad nią nie pracuje (z jej punktu widzenia najważniejsza jest ta funkcja, na której jej zależy). Mając zdefiniowany zestaw zadań, którym przypisane są priorytety, można jej pokazać, nad jakimi funkcjami trwają prace i dlaczego są one najważniejsze. Takie wyjaśnienie z reguły zadowala taką osobę i udowadnia, że wykonywana jest użyteczna praca.

Pełna lista funkcji z przypisanymi sensownie priorytetami zwiększa wzajemne zaufanie w zespole, kierownictwie i innych grupach, które mogą być zależne od naszych funkcji. Posiadanie listy zadań pokazuje, że praca wykonywana jest w sposób zaplanowany.

**▸ WSKAZÓWKA 14**

Pracuj z listą zadań

## Jak korzystać z listy zadań?

Listy zadań możesz używać w pracy własnej lub całego zespołu. Obydwa sposoby są łatwe i efektywne. My korzystamy z niej na oba te sposoby.

Rozpoczęcie korzystania z osobistej listy zadań jest łatwe. Najpierw należy utworzyć listę wszystkich zadań, nad którymi pracujesz (lub które na Ciebie czekają). Następnie przy pomocy kierownika technicznego nadaj wszystkim pozycjom priorytety. Na koniec oszacuj czas potrzebny na zrealizowanie każdej pozycji. Nie przejmuj się, jeśli na początku oszacowanie okaże się nietrafne, z czasem się poprawisz.

Wprowadzenie grupowej listy zadań również nie jest trudne, jeżeli produkt jest dobrze zdefiniowany. Pozwoli ona całemu zespołowi zrozumieć ogólny kierunek, w jakim zmierza projekt.

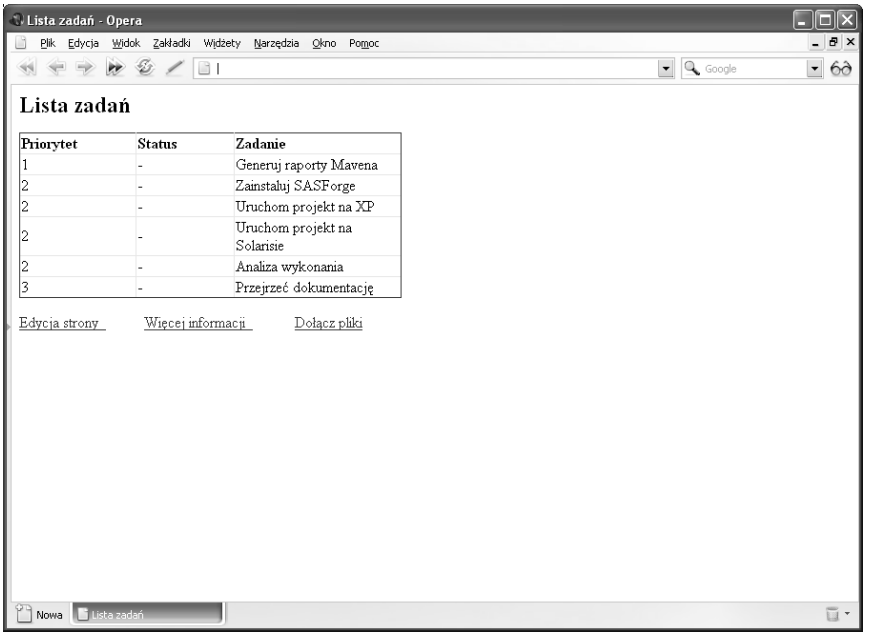
### Lista zadań jako narzędzie organizacyjne

W naszej pracy zawodowej pełniliśmy wiele różnych ról. Z reguły, jak większość ludzi, mieliśmy jednocześnie do wykonania wiele zadań i bardzo mało czasu na ich realizację. Kiedy próbujemy zaplanować sobie pracę nad tymi projektami, na ogół dzielimy swój czas pomiędzy tak wiele zadań, że nie udaje nam się osiągnąć żadnego znaczącego postępu.

Naszym rozwiązaniem jest wykorzystanie listy zadań również do organizacji własnej pracy. Zawsze używamy grupowej listy zadań dla całego zespołu, stwierdziliśmy, że posiadanie własnej kopii zawierającej, co mamy do zrobienia, również jest bardzo pomocne. Każdego ranka zrób listę czekających Cię zadań i nadaj im priorytety. Na koniec dnia przejrzyj, co udało Ci się z niej wykonać, a czego nie. Oceń, czy nie udało Ci się wszystkiego wykonać, ponieważ byłeś zbytnim optymistą, czy też coś w ciągu dnia oderwało Cię od wykonywania zaplanowanych zadań. Pełniejsze omówienie sposobów posługiwania się własną listą zadań można znaleźć w książce Stephena Coveya „Siedem nawyków skutecznego działania".

1. Zapisz na tablicy każdą funkcję, którą dodajesz do projektu. Może to zająć chwilę i często może się okazać, że jedna tablica nie wystarczy.
2. Każdej funkcji nadaj priorytet. Uwzględniaj zdanie odpowiednich osób (kierownictwa, klientów itd.). Idealnie jest, jeśli uczestniczy w tym cały zespół, ale jeśli rzetelnie zebrałeś opinie pracowników, wygodniejsze może być uwzględnienie tylko kierownictwa.
3. Przepisz wszystkie funkcje w kolejności nadanych priorytetów.
4. Dołącz do każdej pozycji prognozowany czas wykonania.

Dopóki nie zostaną zakończone pozycje o najwyższym priorytecie, nikomu nie wolno pracować nad tymi o niższym priorytecie. W ten sposób wszystkie najważniejsze zadania są w toku, zanim rozpoczęte zostanie wykonywanie tych o niższym priorytecie.




**Rysunek 3.2** Lista zadań na stronie intranetowej

Korzystanie z listy zadań jest więc stosunkowo łatwe. Aby jednak zachować efektywność, lista zadań musi spełniać kilka zasad. Musi ona:

- Być ogólnie dostępna.
- Mieć przypisane priorytety.
- Mieć prognozy czasu wykonania.
- Być dynamiczna.
- Być wymierna.
- Być ukierunkowana.

Przyjrzymy się teraz, jakie jest znaczenie każdej z tych reguł i co one oznaczają dla nas oraz dla naszego zespołu.

**Jaś pyta...**

### Co to są kanały RSS?

Kanały RSS są sposobem powiadamiania o zmianach zawartości strony internetowej. Są bardzo popularne na stronach o często zmienianej treści (na przykład serwisy informacyjne lub strona informująca o statusie kompilacji). Strona internetowa z obsługą kanałów RSS generuje listę zmian lub nowych treści w postaci pliku XML.

*Czytnik RSS* jest to program, który sprawdza wszystkie zaprenumerowane kanały RSS i pokazuje nowe treści. Generatory RSS umieszczone są na serwerach WWW, a więc czytnik RSS jedynie sprawdza plik znajdujący się na serwerze i pokazuje zmiany.

Czytnik RSS jest wygodnym narzędziem gromadzenia informacji w zwięzłej formie. Wiadomości są zbierane, aż użytkownik zechce je odczytać.

## Ogólnie dostępna

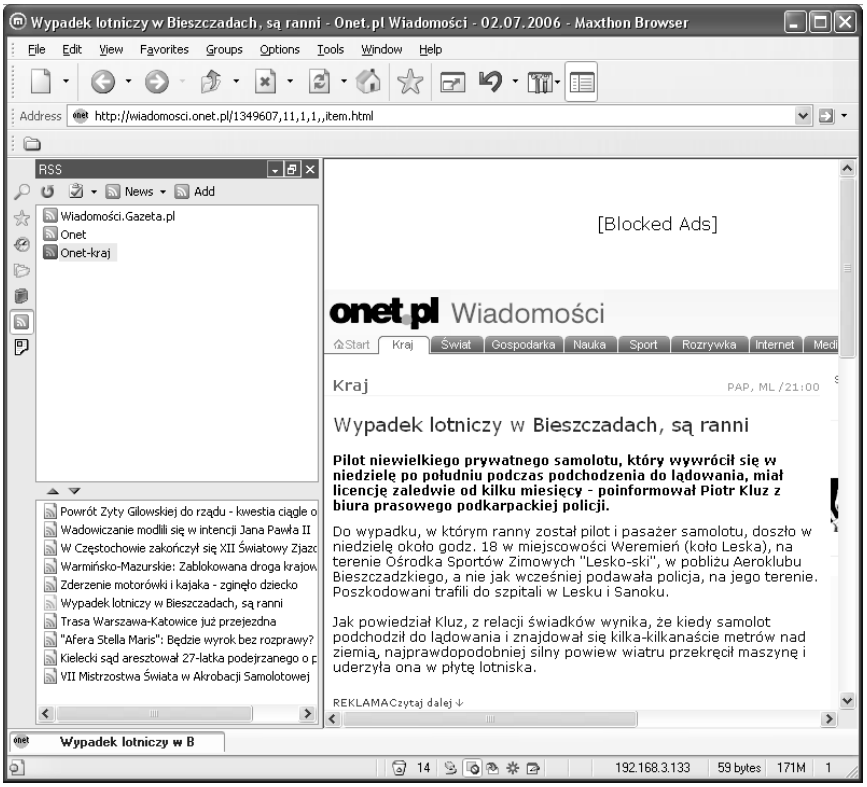
Lista zadań zespołu musi być ogólnie dostępna. Tajna lista nie ułatwia współpracy. Umieść listę zadań na tablicy lub na stronie internetowej, utwórz dla niej kanał RSS. W każdym przypadku powinno być możliwe łatwe jej przeczytanie. Posiadanie listy zadań przed oczami ułatwia zachowanie perspektywy. Daje ona przegląd niewykonanych zadań, który można łatwo prześledzić — zwłaszcza kiedy wykonuje się kilka zadań jednocześnie. Jej publiczna dostępność pozwala również na orientację kierownictwa w bieżących działaniach.

## Przypisane priorytety

Lista zadań musi mieć nadane priorytety. Bardzo ważne jest rozpoznanie różnych rodzajów funkcji związanych z projektem: funkcji niezbędnych, pożądanych i nieistotnych. *Musisz* dokonać tych rozróżnień, przypisując priorytety pozycjom listy — w przeciwnym wypadku tylko stracisz czas. Zawsze

istnieje zestaw podstawowych funkcji, które muszą być zrealizowane, zanim produkt trafi do sprzedaży; są to funkcje o najwyższym priorytecie. Mogłyby one obejmować na przykład ekran logowania, instalator lub bazę danych. Produkt nie może po prostu bez nich działać. Ładniejszy kolor tła w oknie dialogowym *O programie…* prawdopodobnie można uznać za zadanie nieistotne.

Nigdy nie ignoruj nadanych priorytetów. Zakończ wszystkie zadania o wyższym priorytecie, zanim zaczniesz pracę nad mniej ważnymi, chyba że istnieje *istotny* — i przekazany wszystkim — powód, aby tymczasowo zawiesić zadanie z wysokim priorytetem.




**Rysunek 3.3** Czytnik kanałów RSS

## Prognozy czasu wykonania

Do listy zadań zawsze powinien być przypisany harmonogram. Nie musi być on nienaruszalny, ale powinien zawierać prognozę czasu wykonania dla każdej funkcji zawartej na liście. Po zrealizowaniu pozycji i rejestracji jej rzeczywistego czasu wykonania, należy zwrócić uwagę na różnice.

Z czasem prognozy czasu wykonania będą się stawać coraz bardziej dokładne. Po kilku próbach kierownik techniczny powinien być w stanie tworzyć przybliżone harmonogramy projektów na podstawie list zadań poszczególnych członków zespołu. Podobnie powinno być w przypadku kierownika projektu. *Nie istnieją błędne prognozy*. Jedne będą dokładniejsze, inne mniej. Nie martw się, jeśli na początku popełnisz duży błąd. Tej umiejętności nabiera się z czasem[2].

## Dynamiczna

Lista zadań, jeśli ma być efektywna, musi być dynamiczna. Zespół musi być w stanie dostosowywać się do zmian. Kierownik techniczny będzie korygował priorytety funkcji; będą pojawiały się nowe funkcje, a inne będą znikały. Priorytety zmieniają się. I tak powinno być! Zanim się do tego przyzwyczaisz, może być to frustrujące, ale pamiętaj, że Twoja firma stara się być konkurencyjna na zmieniającym się rynku. Również Ty musisz być elastyczny. Zamiast z tym walczyć, staraj się to zaakceptować.

Zmiany listy zadań zwykle oznaczają, że klienci i udziałowcy przyglądają się projektowi i poświęcają mu swoją energię i czas, dając przydatne sprzężenie zwrotne. Większość klientów czeka na koniec projektu i dopiero wtedy mu się przygląda. Wówczas jest już jednak za późno. Zawsze lepiej otrzymywać sprzężenie zwrotne wcześniej, nawet jeśli ciągłe zmiany listy zadań będą frustrujące. Jeśli lista zadań przez jakiś dłuższy czas się nie zmienia, to nie odzwierciedla prawdopodobnie aktualnych priorytetów w projekcie.

---

[2] Jeśli istnieją trudności przy wyznaczaniu prognoz, spróbuj ograniczyć listę możliwości; na przykład każda prognoza czasu wykonania może być jednodniowa, tygodniowa, dwutygodniowa lub czterotygodniowa. Na początek nie dopuszczaj innych możliwości.

## Wymierna

Jeżeli lista zadań ma przynosić efekty, każda pozycja na niej musi być wymierna. W końcu muszą istnieć kryteria pozwalające stwierdzić, że zadanie zostało zrealizowane i może być usunięte z listy.

Kryterium to eliminuje nieprecyzyjne pozycje w rodzaju „Poprawienie wydajności", a promuje takie jak „Skróć czas logowania do poniżej pięciu sekund" lub „Generuj raport X w czasie krótszym niż 10 sekund". Sformułowanie celu, którego stan jest binarny, umożliwia stwierdzenie, czy został on zrealizowany. Cel w rodzaju „Poprawienie wydajności" może trwać przez cały czas życia produktu i pozostać czarną dziurą.

Jeśli masz na liście zadań pozycje, które nie są wymierne, zastanów się nad tym, jakie jest rzeczywiste wymaganie. Czy pozycja wynika z zapotrzebowania na szybsze generowanie raportów czy szybszy start systemu? Podziel tę pozycję na dobrze zdefiniowane, binarne cele, a następnie zapytaj osobę, która wyznaczyła pierwotny cel, o jej zdanie o zmodyfikowanych celach. Ten przegląd upewni Cię, że rzeczywiście bierzesz pod uwagę wymagania klienta.

### Harmonogram według funkcji zamiast harmonogramu według terminów

Harmonogramowanie według terminów ma tę wadę, że klientom sprzedawane są przecież funkcje produktu, a nie dni kalendarzowe.

Jeśli zespół korzysta z listy zadań i wszystkie pozycje są uporządkowane według priorytetów, wówczas prace nad programem podporządkowane są funkcjom, a nie terminom. Kierownictwo firmy może spojrzeć na listę zadań i narysować kreskę pod funkcjami, które chce mieć w następnej wersji. Następnie dodawane są czasy wykonania poszczególnych funkcji i obliczana jest data wyprodukowania nowej wersji. Specyfika produktu i branży zadecyduje, ile czasu należy poświęcić na wewnętrzne testy i wersje beta, ale moment *zamrożenia* prac nad programem zostanie jasno ustalony.

Jeśli dział sprzedaży zdecyduje, że określona funkcja musi być uwzględniona, wówczas jej priorytet na liście zadań może się zmienić i może ona stać się funkcją, która znajdzie się w gotowym produkcie. Czas potrzebny na stworzenie nowej funkcji musi być oczywiście dodany do daty końcowej projektu.

Taki sposób pracy pozwala działowi sprzedaży oraz kierownictwu dostrzec zależność pomiędzy dodawaniem nowych funkcji, a czasem koniecznym na ich zrealizowanie. Nie muszą już podejmować decyzji o terminach realizacji projektu oraz funkcjach produktu, będąc w próżni. Zamiast abstrakcyjnego porównywania dwóch nieokreślonych funkcji, porównują dwie jasno zdefiniowane z określonymi czasami realizacji.

Uważamy, że dzięki temu podejściu produkt dostępny jest wówczas, gdy jest gotowy, tak szybko jak to możliwe. Unika się również arbitralnego wyznaczania terminów, które nie są przestrzegane. Nasza branża słynie z niedotrzymywania terminów i nic dziwnego, jeśli weźmiemy pod uwagę sposoby, w jakie piszemy programy. Zamiast na siłę zakładać arbitralny termin na arbitralny zbiór funkcji, firmy powinny pytać swoich programistów, co są w stanie wykonać! Jeśli programiści nie są w stanie wykonać zadania, to czy lepiej jest dowiedzieć się o tym teraz czy później, kiedy termin zostanie już przekroczony? A jeśli programiści są w stanie wykonać zadanie *przed* terminem, czy nie byłoby dobrze wiedzieć o tym i dodać ewentualnie nowe funkcje lub wysłać program klientowi wcześniej?

Za pierwszym razem kiedy będziesz kończył produkt w opisany sposób, kierownictwo będzie zaniepokojone. Za drugim razem, będą odprężeni. Za trzecim będą mieli zaufanie, że ich programiści dostarczą to, co obiecali!

Jeżeli pozycja nie może być wyrażona za pomocą wymiernych celów, wówczas nadaj jej najniższy priorytet i zajmij się tymi o wyższym priorytecie. Całkowite usunięcie pozycji mogłoby być błędem, jeśli była ona jednak istotna; pozycja musi być jednak przekształcona w wymierne części.

**Jaś pyta...**

### Co to jest zamrożenie kodu?

*Zamrożenie kodu* to moment, od którego kod programu nie może już podlegać zmianom. W trakcie trwania projektu kod programu podlega nieustannym zmianom. Po jego zamrożeniu zmiany zostają zatrzymane, tylko poważne poprawki błędów mogą być wtedy wykonane. Nowe funkcje oraz poprawki drobnych błędów nie są dopuszczalne.

Często zamrożenie kodu przechodzi w odwilż, w miarę jak nieprzemyślane zmiany przedostają się do programu.

## Ukierunkowana

Zapewne zauważyłeś, że omawialiśmy zarówno indywidualne listy zadań, jak i grupowe. Każdy rodzaj listy jest bardzo ważny i musi być w swojej treści odpowiednio ukierunkowany. Lista zadań dla całego zespołu będzie dużo bardziej rozbudowana i będzie zawierać wszystkie niewykonane zadania dla całego projektu. Twoja osobista lista zadań będzie zawierała mniej pozycji związanych z zespołem (czasem tylko jedną pozycję z całego projektu), ale jak tylko ją zrealizujesz, przeniesiesz pozycję z listy grupowej na swoją własną.

Mimo swojej prostoty lista zadań jest skutecznym narzędziem na wielu szczeblach. Poprawia Twoją organizację pracy i zwiększa dostęp do informacji dla kierowników projektu. Tworzenie i nadawanie priorytetów pozycjom listy zadań wymaga przemyślenia własnej pracy i nakreślenia kolejnych kroków. Każdy wielki bilardzista powie Ci, że ma zaplanowane następne osiem uderzeń — podobnie jest z wielkim programistą.

## Przykład listy zadań

Oto przykład indywidualnej listy zadań uporządkowanej według priorytetów:

1. Dodaj nowy raport wyświetlający formanty utworzone w ciągu dnia.
2. Dodaj nowy raport wyświetlający formanty utworzone przez pracownika.
3. Sprawdź błąd #12345 (raport miesięczny wskazuje, że liczba utworzonych formantów wynosi zero).
4. Zainstaluj narzędzia programistyczne na swoim nowym komputerze.
5. Sprawdź nowe formaty raportów. Mogą one być dobrym dodatkiem do następnej wersji.

Zauważ, że najważniejsze pozycje są na pierwszych miejscach. W tym przypadku nowe funkcje są ważniejsze niż poprawki błędów, ale nie zawsze tak jest. Ponadto instalacja nowego komputera i projekt badawczy są u dołu listy. Te pozycje mogą wypełnić czas w wolnych chwilach (na przykład gdy czekasz na kogoś) lub gdy potrzebujesz przerwy. Ważne jest umieszczanie na liście zadań o niskim priorytecie, aby uniknąć ich zapomnienia.

## Od czego zacząć?

1. Przez cały dzień zapisz każde zadanie, nad którym pracujesz (to będzie lista zadań zakończonych).
2. Zapisz swoje codzienne działania w sformalizowany egzemplarz listy zadań.
3. Poproś kierownika technicznego o pomoc w nadaniu priorytetów i dodaj wstępne prognozy czasu wykonania.
4. Rozpocznij pracę od pozycji listy o najwyższym priorytecie — bez oszustw! Jeśli jakiś kryzys zmusi Cię do zajęcia się czynnością o niższym priorytecie, zarejestruj to.

5. Dodaj wszystkie nowe zadania do listy.
6. Przenoś pozycje na listę czynności zakończonych po zrealizowaniu zadania (dzięki temu łatwiej przeżyć raporty o statusach i „polowania na czarownice").

Tworzenie listy zadań zmusza do organizacji swojej pracy i nadania jej priorytetów. Tak samo jak pisanie pamiętnika pomaga przemyśleć i zrozumieć, co się robiło. Lista pozwala uporządkować własne zadania na dość wysokim poziomie i bez zbędnej formalizacji.

Przeglądaj listę zadań każdego ranka. Aktualizuj ją, kiedykolwiek pojawiają się nowe zadania... zwłaszcza krytyczne, pojawiające się w ostatniej chwili — mógłbyś o nich zapomnieć, gdy ktoś zapyta, co robiłeś przez cały ubiegły tydzień.

## Postępujesz prawidłowo, jeśli...

- Czy każde Twoje bieżące zadanie znajduje się na liście?
- Czy lista zadań poprawnie odzwierciedla Twoje aktualne obowiązki?
- Czy kierownik techniczny lub klient pomagał Ci nadawać pozycjom listy priorytety?
- Czy lista jest ogólnie dostępna (elektronicznie lub w inny sposób)?
- Czy skorzystałeś z listy, aby wybrać swoje kolejne zadanie?
- Czy potrafisz szybko aktualizować i udostępniać listę?

## Sygnały ostrzegawcze

- Nie dodajesz zadań do listy, ponieważ jesteś „zbyt zajęty".
- Aktualizacja listy zajmuje więcej czasu niż realizacja zadań.
- Realizacja pozycji listy przez członków zespołu ciągnie się tygodniami (wskazówka: pozycje są zbyt rozbudowane).

- Lista zadań jest aktualizowana rzadziej niż raz w tygodniu.
- Priorytety na liście nie odpowiadają rzeczywistym priorytetom.
- Lista zadań jest trzymana w ścisłej tajemnicy i nie jest widoczna dla osób spoza zespołu.
- Poza listą zadań zespołu istnieją inne ogólnie dostępne wersje (które różnią się między sobą).

**Jaś pyta...**

**A jeśli mój zespół nie chce korzystać z listy zadań?**

Nawet jeśli Twój zespół nie korzysta z listy zadań, możesz posługiwać się nią we własnym zakresie. Zapisz ją na tablicy lub w notesie, w palmtopie lub na swojej stronie internetowej. Poproś kierownika technicznego o pomoc w ustaleniu priorytetów i wykorzystaj jego rady do uporządkowania swojej pracy. Zaznaczaj, które pozycje już wykonałeś, i przeglądaj listę co tydzień. Korzystanie z listy w taki sposób będzie dla Ciebie równie efektywne, jak byłoby efektywne dla całego zespołu. W końcu dobry kierownik techniczny zauważy, że korzystasz z listy zadań i zacznie jej używać dla całego zespołu. Będziesz zaskoczony, jak szybko dobre nawyki stają się popularne. Może to chwilę potrwać, ale ludzie zawsze w końcu przekonują się do czegoś, co jest skuteczne.

# 11. Kierownik techniczny

*kierownik techniczny*

*Kierownik techniczny* ponosi techniczną odpowiedzialność za projekt programistyczny. Obecność kierownika technicznego zwalnia menedżera firmy od technicznych obowiązków na rzecz kogoś z lepszymi predyspozycjami, on sam może natomiast zająć się innymi biurokratycznymi sprawami. Menedżer może być jednocześnie kierownikiem technicznym, ale nie jest to konieczne, a często wręcz niepożądane. Obecność oddzielnego kierownika technicznego jest wskazana, jeśli menedżer nie ma dostatecznej wiedzy technicznej lub jeśli zespół pracuje jednocześnie nad kilkoma projektami.

## Dlaczego potrzebny jest kierownik techniczny?

Czy zdarzyło Ci się pracować pod kierownictwem menedżera, który nie rozumiał używanej przez Ciebie technologii? Tego typu osoby ustalają nierealne terminy, nie rozumieją opóźnień i narzekają na harmonogramy. Wszystko to dzieje się dlatego, iż nie rozumieją tego, co robisz i jak funkcjonują wykorzystywane przez Ciebie technologie. Dla osoby, która nigdy nie pracowała jako programista, zrozumienie Twojej pracy będzie trudne. Podczas gdy w dziale sprzedaży mogą pracować osoby, które wiedzą tylko, jak korzystać z Twojego programu, w przypadku kierownika technicznego wymagana jest dokładna wiedza na temat tego, jak ten program funkcjonuje. Osoba odpowiedzialna za harmonogramowanie funkcji i koordynowanie pracy programistów, aby być efektywna, musi rozumieć, jak działa kod programu. Potrzebujesz osoby biegłej technicznie, która będzie w stanie wyjaśnić technologie związane z produktem menedżerom bez wiedzy technicznej. Potrzebujesz osoby, która będzie pośrednikiem pomiędzy zespołem programistów a zarządem firmy. Potrzebujesz kierownika technicznego.

Na ogół kierownik techniczny jest członkiem zespołu, który awansuje na stanowisko kierownicze. Rozumie on techniczne problemy zespołu dzięki swojemu doświadczeniu w programowaniu. Nie zobowiązuje się do nierealnych terminów, ponieważ rozumie, czego wymaga pozornie prosta nowa funkcja. Kierownik techniczny może wyeliminować, a przynajmniej zmniejszyć, wtrącanie się kierowników bez odpowiednich kompetencji.

Po jednej stronie znajduje się menedżer bez kompetencji technicznych, po drugiej taki, który cały dzień pogrążony jest w pisaniu programów i nie kontaktuje się ani z kierownictwem, ani z klientami. Jest to stereotypowy programista ukrywający się w swoim biurze i unikający pozatechnicznego świata. Ale wówczas firma opracowuje programy, których nikt nie chce, a priorytety wynikają raczej z kaprysów menedżera, a nie z potrzeb klientów. Za każdym razem powstaje produkt, który zalega na półkach. Kierownicy wyższych szczebli nie wiedzą, czym Twój zespół się zajmuje, a więc zakładają, że nie robi nic, to z kolei niweczy Twoje szanse na premie, podwyżki i awanse. Twój zespół potrzebuje adwokata, który znajdzie czas

na pisanie raportów i przygotowywanie prezentacji dla menedżerów, klientów i wszystkich osób, dla jakich znajomość Twojego projektu jest istotna. A więc jeszcze raz: Twój zespół potrzebuje kierownika technicznego.

Kierownik techniczny żyje więc w dwóch światach. Musi pracować wspólnie z programistami i ich rozumieć, a jednocześnie spotykać się z menedżerami, klientami i innymi kierownikami technicznymi, aby przekazywać informacje o pracy własnego zespołu. Poświęcając czas na spotkania z klientami, jest jedyną osobą, która wie, jakie funkcje powinien wykonywać tworzony program.

Kierownik techniczny spełnia następujące funkcje:

- Zapewnia, że priorytety zespołu odpowiadają potrzebom klienta.
- Odpowiednio przedstawia wykonywane w zespole zadania kierownictwu firmy.
- Izoluje programistów od menedżerów wyższego szczebla, którzy nie mają przygotowania technicznego.
- Tłumaczy techniczne kwestie udziałowcom bez odpowiedniego przygotowania.
- Przekazuje pozatechniczne problemy programistom.

**Jaś pyta...**

### Kim jest udziałowiec?

*Udziałowiec* jest to osoba, która ma związek z Twoim produktem. Pracując nad projektem, powinieneś wiedzieć, kim będą jego odbiorcy. Twoimi udziałowcami mogą być klienci, ale jeśli ich jeszcze nie poznałeś, może ich zastąpić dział sprzedaży lub marketingu. W małych firmach udziałowcami mogą być inwestorzy. W większych firmach udziałowcami są czasem inne działy, które rozwijają swoje produkty, opierając się na Twoich. Jest bardzo istotne, aby rozpoznać te osoby tak, aby stworzyć produkt, którego potrzebują.

Jakie więc czynności musi wykonać kierownik techniczny, aby zrealizować te zadania?

## Zakres odpowiedzialności kierownika technicznego

Kierownik techniczny ma kilka ważnych obszarów odpowiedzialności, które różnić się będą w zależności od rodzaju firmy i składu zespołu. Oto minimalna lista kompetencji kierownika technicznego:

- ustalanie wytycznych dla programistów,
- zarządzanie listą funkcji projektu,
- ustalanie priorytetów,
- izolowanie członków zespołów od czynników zewnętrznych[3].

Przyjrzyjmy się bliżej każdej z tych dziedzin.

## Ustalanie wytycznych dla programistów

Kierownik techniczny jest reżyserem zespołu, ustala kierunki i priorytety. Współpracuje z każdym programistą przy tworzeniu i modyfikowaniu osobistej wersji listy zadań (patrz: podrozdział 10. na stronie 78). Kierownicy techniczni korzystają ze swojej znajomości postępów, problemów oraz prognozowanych terminów realizacji związanych z ich zespołem, tak aby wytworzyć ogólny obraz stanu projektu oraz śledzić jego postępy. Jako pojedynczy punkt kontaktowy dają oni możliwość szybkiego powiadamiania o statusie projektu dla wszystkich zainteresowanych.

## Zarządzanie listą funkcji projektu

Kierownik techniczny jest głównym nadzorcą listy funkcji związanych z projektem. Zapotrzebowanie na nowe funkcje nie jest przekazywane bezpośrednio programistom, tylko wszystkie są najpierw zgłaszane do kierownika technicznego. Zarządza on wszystkimi zmianami na liście funkcjonalności.

[3] Istnieje ryzyko przesadnej izolacji, która może pozbawić wartościowych sprzężeń zwrotnych.

Kierownik techniczny jest jedyną osobą, która rozumie nie tylko powiązania techniczne każdej funkcji, ale zna również życzenia udziałowców projektu. Kierownik techniczny dodaje i usuwa funkcje oraz ukierunkowuje wysiłki poszczególnych programistów.

Kierownik techniczny oraz udziałowcy najpierw ustalają zakres pracy poprzez stworzenie listy funkcji (najlepiej za pomocą listy zadań!). Następnie kierownik spotyka się z całym zespołem, aby oszacować czas potrzebny na realizację każdej funkcji. Czasem oznacza to podział rozbudowanej funkcji na kilka pozycji listy zadań. Następnie przypisuje on poszczególne zadania programistom i ustala wstępny harmonogram projektu (wstępne oszacowania podlegają zmianom w miarę postępów projektu, ale więcej na ten temat później).

Zarządzanie listą funkcji przez kierownika technicznego bardzo pomaga w sytuacji, gdy ciągle pojawiają się osoby próbujące dodać do projektu nowe funkcje. Kierownik techniczny pełni rolę bufora filtrującego zapotrzebowania i ustalającego dla nich rozsądne priorytety.

Kierownik techniczny może być bardzo przydatny jako pośrednik w kontaktach z zespołem programistów. Na przykład w wielu firmach spotyka się kilku nadmiernie „kreatywnych" pracowników. Mają oni zawsze wspaniałe pomysły, ale rozpraszają tym innych pracowników. Ten typ pracowników ma zwyczaj zjawiać się, kiedy inni są najbardziej zajęci.

Jeden szczególnie kreatywny pracownik (Ernest), którego znaliśmy, był gorszy niż inni. Aby zniwelować jego wpływ, kazaliśmy mu przynosić pomysły do nas (a nie do programistów). Następnie dodawaliśmy jego pomysł na nową funkcję do listy zadań (która znajdowała się na tablicy). Po jej dodaniu porównywaliśmy ją z funkcjami o priorytecie pierwszym (wymagane funkcje) i wszyscy zgadzaliśmy się, że nie zasługuje ona na taki priorytet. Następnie dokonywaliśmy kolejno porównań z funkcjami o coraz niższym priorytecie. Okazywało się, że zawsze kończyliśmy na najniższym priorytecie — piątym, związanym z opcjonalnymi, mało istotnymi funkcjami.

Ostatecznie Ernest poznał nasz system i zaczął sam dodawać swoje funkcje na dole tablicy, poniżej tych o priorytecie piątym. Ponieważ umieszczał je na liście zadań, jego funkcje nie były zapominane. Niektóre ostatecznie zostawały dodane do naszego produktu, ale zawsze umieszczane były w następnej wersji, a nie w bieżącej. Postępując tak z kreatywnymi osobnikami, dajemy im możliwość zgłaszania swoich pomysłów, a zespół nie jest odrywany od bieżącej pracy i nie traci czasu na nieistotne funkcje.

## Ustalanie priorytetów

Tworzenie nowej listy zadań powinno być uporządkowane. Bez tego każdy mógłby po prostu wybrać interesujące go funkcje, a zaniedbać te, które są konieczne. Określenie poprawnego priorytetu dla każdej z funkcji jest niemal tak samo ważne jak samo umieszczenie pozycji na liście! Na szczęście jest to problem, który możesz zlecić swojemu kierownikowi technicznemu. Musi on współpracować z udziałowcami projektu, aby ustalić priorytety funkcji.

Udziałowcy mają duży wpływ na priorytety funkcji i zawsze wykazują szczególne zainteresowanie projektem oraz jego powodzeniem. Nie mają oni jednak na ogół przygotowania technicznego niezbędnego do podejmowania właściwych decyzji. Często po prostu nie wiedzą, co jest technicznie możliwe oraz praktyczne.

Kierownik techniczny współpracuje z udziałowcami, aby ustalać priorytety funkcji. Może on wykonać tę czynność, ponieważ zna zdolności swojego zespołu (pojęcie, które nieustannie podlega zmianom), a także techniczne szczegóły projektu.

W trakcie tego procesu kierownik techniczny pozna pozatechniczne powody, dla których udziałowcy domagają się pewnej funkcji. Kierownik techniczny oraz udziałowiec pracują razem i osiągają kompromis przy ustalaniu priorytetów. Ta współpraca pomaga wyważyć racje udziałowca bez przygotowania technicznego i kierownika technicznego, który nie ma kontaktów z klientem.

W praktyce kierownik techniczny nie musi się spotykać z udziałowcem, aby dyskutować o każdym detalu. Z czasem zaczną się nawzajem rozumieć i sobie ufać.

## Izolowanie programistów od czynników zewnętrznych

### Priorytety

Z priorytetami skojarzone są liczby. Najwyższy jest priorytet pierwszy, najniższy piąty. W razie potrzeby można stosować inną numerację. Obecnie korzystamy z zakresu od jeden do pięć, ale w innych sytuacjach stosowaliśmy zakres od jeden do dziesięć. Istotniejsze jest uporządkowanie, a nie zakres.

- Priorytet pierwszy: funkcje wymagane.
- Dotyczy funkcji, bez których absolutnie nie można sprzedać produktu.
- Priorytet drugi: funkcje bardzo ważne.
- Można by sprzedać produkt bez zawarcia tych funkcji, ale jest to niewskazane.
- Priorytet trzeci: funkcje przydatne.
- Jeśli znajdziesz czas, zrealizujesz je, ale na pewno nie opóźnią one daty zakończenia projektu.
- Priorytet czwarty: funkcje dekoracyjne.
- Funkcje te poprawiają odbiór programu.
- Priorytet piąty: funkcje nieistotne.
- Jeżeli masz czas na dodawanie funkcji nieistotnych, to znaczy, że jesteś przed terminem i masz wolny budżet.

Jesteś w środku zawiłego projektu. Szło Ci świetnie przez cały poranek, dokonywałeś znacznych postępów. Wtedy natrętny pracownik działu sprzedaży wpada, aby zadać pytanie dotyczące następnej wersji, czym kompletnie Cię rozprasza. Złościsz się, nawet gdy tylko o tym czytasz? Nie

tylko Ty — każdemu pracuje się lepiej, jeśli nikt mu nie przerywa. Badania wykazały, że do 40% czasu pracy może być marnowane wskutek przerywania[4]. To tak jakbyś szedł do domu po przepracowaniu mniej niż pięciu godzin! Naukowcy mają nawet swoje określenie na to zjawisko: *przeciążenie poznawcze*[5]. Wiedząc o tym, kierownik techniczny musi dokładać wszelkich starań, aby pozwalać pracować swojemu zespołowi bez zakłóceń. Świetną metodą jest wykorzystanie kierownika technicznego jako punktu kontaktowego dla programistów. Zawsze pozwalaj kierownikowi technicznemu buforować przerwania, niezależnie od tego, czy pochodzą z działu IT czy od udziałowców.

*przeciążenie poznawcze*

## Skąd pochodzą pozycje listy zadań?

Z jakich źródeł zbierane są funkcje umieszczane na liście zadań? Czy można użyć dokumentacji z analizy wymagań? Analiz wymagań z innych projektów? Historii użytkowników? Kartek z notesu? Dużego rządowego kontraktu? Nie ma to znaczenia. Ważne jest, aby znać funkcje, które można umieścić na liście zadań i nadać im priorytety. Lista zadań nie zastępuje Twojej preferowanej metody zbierania wymagań. Zamiast tego gromadzi informacje z każdego źródła i przedstawia je w precyzyjnym i zrozumiałym formacie.

Jeszcze raz wyjaśnijmy to precyzyjnie. Lista zadań nie zastępuje *sposobu* gromadzenia wymagań. Zastępuje ich sposób przedstawienia Twojemu zespołowi i współpracownikom.

## Jak wygląda Twój kierownik techniczny?

Kierownicy techniczni dzielą swój czas pomiędzy zadania programistyczne i zarządcze. Będą zdarzały się dni, a nawet tygodnie, w których kierownik techniczny będzie działał tylko w jednym z tych zakresów, ale większość dni zostanie pomiędzy nie podzielona.

---

[4] Badania potwierdziły, że zmuszanie pracownika do *zmiany kontekstu* może mieć poważny wpływ na wydajność. Koszt waha się od 20% do 40% — *http://www.umich.edu/~bcalab/multitasking.html*.

[5] Dr David Levy bada, w jaki sposób jednoczesne wykonywanie kilku zadań wpływa na nasze zdrowie i wydajność: *http://seattletimes.nwsource.com/pacificnw/2004/1128/cover.html*.

Kierownik techniczny jest strażnikiem projektu. Kierownicy techniczni utrzymują pracowników na wytyczonym szlaku oraz zapewniają sensowność listy funkcji. Informują również kierownictwo o postępach zespołu i upewniają się, że punkt widzenia klienta jest brany pod uwagę. Twój kierownik techniczny musi być prawdziwym omnibusem.

Oczywiście szerokie kompetencje kierownika technicznego mogą przyczynić się zarówno do sukcesu, jak i niepowodzenia projektu (lub jego części). To nie jest łatwa praca; wymaga przygotowania technicznego, zdolności komunikacyjnych oraz umiejętności równoczesnego wykonywania kilku czynności.

Każdy projekt powinien mieć dobrego kierownika technicznego i każdy programista powinien starać się nim zostać przynajmniej raz. Nawet jeżeli nie będziesz pełnił tej funkcji na stałe, nauczenie się jego umiejętności spowoduje, że staniesz się bezcenny dla swojego zespołu i firmy.

**▸ WSKAZÓWKA 15**

Stwórz stanowisko kierownika technicznego

## Od czego zacząć?

Jeśli masz aspiracje, aby zostać kierownikiem technicznym, musisz udowodnić, że jesteś w stanie radzić sobie z dodatkową odpowiedzialnością. Zapoznaj się z zakresem obowiązków i próbuj im sprostać. Dobrowolnie wykonuj tak wiele obowiązków kierownika technicznego, ile potrafisz. Nie czekaj, aż ta praca spadnie Ci z nieba; pokaż, że starasz się o to stanowisko i potrafisz sobie na nim radzić.

Korzystaj z listy zadań (patrz: podrozdział 10. na stronie 78) do swojej własnej pracy, ale także prowadź ją dla swojego zespołu. Monitoruj pracę bieżącą, jednocześnie pamiętając o zbliżających się projektach.

Oceniaj procesy realizowane w zespole. Lokalizowanie słabych punktów oraz poszukiwanie standardów i pomysłów rozwiązujących problemy postawi Cię w nowej perspektywie. Jeśli na przykład istnieje problem z zapewnieniem poprawnej kompilacji kodu, powinieneś zainstalować system CI (patrz: podrozdział 4. na stronie 49), aby rozwiązać ten problem.

Nie rezygnuj, jeśli nie awansujesz na stanowisko kierownika technicznego natychmiast. Ucz się i rozwijaj, aż pojawi się następna okazja. Nie każdy ma predyspozycje, aby zostać kierownikiem technicznym, ale zmierzanie do tego daje szerszy obraz całego projektu, co czyni Cię bardziej efektywnym pracownikiem. Stajesz się lepszym programistą, myśląc o tym, jak zostać kierownikiem technicznym.

Jeśli właśnie zostałeś kierownikiem technicznym, przygotuj szkic mapy drogowej — wykres pokazujący, w którym miejscu aktualnie znajduje się zespół oraz w jakim kierunku chcesz, żeby zmierzał. Jakimi problemami się zajmiesz? Do jakich prac będziesz zachęcał?

Wykonaj listę wszystkich znanych problemów. Następnie przepytaj pracowników, czy znane są im inne problemy. Kiedy uznasz, że lista jest gotowa, zdecyduj, którymi pozycjami możesz się zająć, a którymi nie.

Codzienne spotkania są znakomitym sposobem nadzorowania prac zespołu bez niepotrzebnego nękania pracowników (patrz: podrozdział 12. na stronie 103).

## Postępujesz prawidłowo, jeśli...

Jako kierownik techniczny powinieneś być w stanie pozytywnie odpowiedzieć na poniższe pytania:

- Czy wiesz, nad czym pracuje każdy pracownik w Twoim zespole?
- Czy potrafisz wygenerować podsumowanie statusu projektu w czasie krótszym niż 5 minut?
- Jakie jest następne 5 – 10 funkcji produktu?
- Czy potrafisz szybko wymienić usterki o najwyższym priorytecie?

- Jaki ostatni problem został rozwiązany przez Twojego pracownika?
- Czy pracownik przyszedłby do Ciebie, jeśli potrzebowałby rozstrzygnięcia jakiejś ważnej kwestii?

### Sygnały ostrzegawcze

Oto kilka sygnałów ostrzegawczych świadczących o tym, że kierownik techniczny jest nieefektywny lub zbyt apodyktyczny:

- Nie rozumie kontekstu pracy każdego z pracowników.
- Swoją obecnością przerywa pracę innych.
- Przypisuje sobie pracę zespołu.
- Nie rozwiązuje problemów lub — co gorsza — je stwarza.
- Niedokładnie prognozuje czas pracy.
- Nie jest świadomy technicznych umiejętności swoich pracowników lub (i) czego oni chcą się nauczyć.
- Nie zwraca uwagi na konflikty personalne.

## 12. Codzienna koordynacja i komunikacja

Ostatnią rzeczą, której pracownicy oczekują, jest większa liczba spotkań. Jeśli Twój zespół spotyka się co tydzień, wówczas każde spotkanie trwa prawdopodobnie przynajmniej godzinę. Obejmuje ono omawianie ogłoszeń, ubiegłotygodniowej pracy i planów na następny tydzień. Codzienne spotkania są zupełnie inne. Są to krótkie spotkania zespołu, które zachęcają do kontaktów i komunikacji bez niepotrzebnego rozbijania codziennego harmonogramu pracy.

Każdy pracownik krótko opowiada, nad czym pracuje i na jakie natknął się problemy. Dobrą regułą jest, aby poświęcać nie więcej niż jedną – dwie minuty na pracownika. Pamiętaj, że to spotkanie angażuje każdego pracownika, a więc powinno być krótkie i na temat, tak aby nie angażować dużych nakładów gotówkowych (ang. *burn rate*).

## Do czego potrzebne są codzienne spotkania?

Najprostszym sposobem radzenia sobie z zakłóceniami w komunikacji jest częstsze rozmawianie z pracownikami. To pomoże zrozumieć, gdzie leży problem, i daje szansę naprawienia go.

Większość ludzi stara się realizować cele zespołu. Niestety ludzie mają skłonność do nierozumienia celów lub zatracania kierunku, co oznacza, że należy podejmować czynności korygujące.

Spotykaj się częściej ze swoim zespołem i pozwól wszystkim opowiadać o swojej pracy. Celem jest dokonywanie częstych korekt kursu: jeśli chcesz jechać samochodem, wybierasz kierunek, ale nie trzymasz się go bez względu na wszystko. Kierujesz samochód na właściwy kierunek, a następnie wykonujesz mnóstwo drobnych korekt, kręcąc kierownicą w prawo lub w lewo. Zjedziesz z trasy i rozbijesz samochód, jeśli będziesz czekał zbyt długo na korektę. Twój projekt programistyczny również skończy się fiaskiem, jeśli zbyt długo nie będziesz przeprowadzał żadnych korekt.

**▸ WSKAZÓWKA 16**

Dokonuj częstych korekt kursu poprzez codzienne spotkania

## Co dają codzienne spotkania?

Codzienne spotkania dają wiele korzyści, których możesz zacząć doświadczać od pierwszego dnia, gdy zaczniesz je stosować.

### Jaś pyta...

#### Czym jest wskaźnik nakładów gotówkowych (ang. burn rate)?

*Wskaźnik nakładów gotówkowych* opisuje, ile kosztuje bieżąca działalność firmy, a więc wynagrodzenia, czynsz, elektryczność, świadczenia itd. Jest to kwota pieniędzy wydawana niezależnie od tego, czy praca jest wykonywana, czy nie jest. Jeśli przeprowadzasz jakieś spotkanie, zawsze cofnij się o krok i oblicz, ile Cię ono kosztuje na godzinę. Jeśli to policzysz, spotkania będą na ogół krótsze.

Niech na przykład praca programisty kosztuje 100 zł na godzinę (po uwzględnieniu pensji oraz kosztów pośrednich). Jeśli w zespole pracuje 10 osób, wskaźnik nakładów gotówkowych wynosi 1000 zł na godzinę, 8000 zł na dzień i 40 000 zł na tydzień. Następnym razem gdy będziesz organizował spotkanie i zacznie się ono z dziesięciominutowym opóźnieniem lub wszyscy przez 30 minut będą słuchać opowieści o wakacjach Marka, oblicz, jakie są koszty. Jeśli Twój produkt ma trzy miesiące spóźnienia, jakie były w tym czasie koszty prac programistycznych? Pamiętaj, że ten przykład nie bierze pod uwagę kosztów utraconych korzyści podczas zmarnowanego czasu.

## Zeszliśmy z kursu... znowu

Istnieją programiści, którzy wiecznie schodzą z kursu. Najczęściej są to programiści niedoświadczeni oraz indywidualiści. Nowi pracownicy nie mają doświadczenia i będą spędzali czas nad rozwiązywaniem problemów dawno rozwiązanych. Indywidualiści są starsi i bardziej doświadczeni, ale schodzą z kursu równie często. Codzienne spotkania pozwalają obie te grupy programistów utrzymać na właściwym kursie.

## Wyważanie otwartych drzwi

Niedoświadczony programista będzie rozwiązywał problemy, które nie potrzebują rozwiązania. Bystrzy młodzi programiści rzadko spotykają problemy, których nie potrafią rozwiązać, ale te, które rozwiązują, zwykle zostały już rozwiązane przez kogoś innego. Nieszczęśliwie dla nich, inni programiści nie rozmawiają z nimi wystarczająco dużo, aby im o tym powiedzieć. Zamiast uczyć się na błędach swoich współpracowników, nowi programiści wyważają otwarte drzwi i nie robią zbyt dużych postępów.

Zauważysz ten problem, gdy programiści zaczną wykorzystywać struktury danych języka programowania. Ci programiści tworzą sprytne formanty GUI, które powielają to, co przygotował inny programista miesiąc temu. Każdy w firmie pisze własną kopię każdego narzędzia. Jeśli istniałaby komunikacja, wówczas narzędzia te można by pisać jednokrotnie. Jednak programiści w tego typu firmach są zbyt zajęci, aby ze sobą rozmawiać. W efekcie każdy z nich ma formanty GUI o nieco innym wyglądzie i z własnymi procedurami obsługi łańcuchów znaków.

Gdy Twój zespół spotyka się codziennie, a Wasz nowy pracownik Marek mówi wszystkim, że zaczyna opracowywać nowy zestaw formantów GUI, Agnieszka może mu powiedzieć o formantach, nad którymi ona i Michał pracowali w ubiegłym roku. Być może nie są one dokładnie tym, czego on potrzebuje, ale stanowią dobry punkt wyjścia dla jego pracy. Zamiast pracować od zera, Marek będzie korzystał z pracy już wykonanej przez jego kolegów. Jeśli prowadzimy codzienne spotkania, wówczas mamy forum dla tego rodzaju dyskusji. Nie musimy czekać, aż Marek przez przypadek wspomni o swoich formantach GUI podczas obiadu albo przerwy na kawę.

## Indywidualiści

Wszyscy kiedyś pracowaliśmy z programistami-indywidualistami. Mają oni skłonność zatracać kierunek i dryfować, nie wiadomo gdzie. Brną pracowicie przez przypadkowo wybrany kod i „poprawiają" go poprzez porządkowanie nagłówków metod, szlifowanie algorytmów i formatowanie nawiasów. Indywidualiści nie mają wystarczająco dużo dyscypliny, aby zakończyć jakiekolwiek zlecone zadanie i na ogół przynoszą więcej szkody niż pożytku.

Jeden programista-indywidualista, którego znamy, porządkuje nagłówki metod i usuwa nieużywane argumenty. Przykładowo w jego rękach wiersz

```
doSomething(String foo, Integer bar)
```

przekształciłby się w

```
doSomething(String foo)
```

Niestety, ten programista, podobnie jak inni indywidualiści, nigdy równocześnie nie zmienia kodu, który używa metody `doSomething`, wskutek czego jego zmiany często powodują niepowodzenia kompilacji.

Indywidualiści będą poprawiać wydajność algorytmów, ale ich zmiany spowodują, że program zacznie generować błędne wyniki. Będą spędzać godziny na formatowaniu programu tak, aby odpowiadał ich gustom. Indywidualiści mają zbiór nawyków, które stosują bo „tak trzeba".

Nad indywidualistami można zapanować dosyć łatwo. Po pierwsze: przeprowadzaj codzienne spotkania. Po drugie: zwracaj uwagę na codzienne raporty indywidualistów. W tym miejscu dobry kierownik techniczny zauważy ich dryfowanie, zanim zajdzie ono za daleko. I na koniec: kierownik techniczny musi mieć pewność, że ich praca jest warta całego dnia spędzonego w biurze. Taka taktyka nie eliminuje całkowicie ich błąkania się, ale zminimalizuje ich wolny czas i ograniczy szkody.

## Przekazywanie wiedzy

Jeśli w zespole jest kilku doświadczonych programistów, codzienne spotkania pozwalają skorzystać z ich wiedzy. Za każdym razem, gdy dzielą się swoim rozwiązaniem, dowiaduje się o nim cały zespół. Z drugiej strony za każdym razem, gdy młodszy pracownik opowiada o problemie, starsi dowiadują się o nim i mogą pomóc — jeśli nie wiesz o problemie, nie możesz pomóc go rozwiązać.

Powiedzmy na przykład, że Jarek przychodzi na codzienne spotkanie i mówi, że ma problem z parserem XML-a, który nie radzi sobie ze znakami międzynarodowymi. Michał, który spotkał się z takim samym problemem rok temu, może powiedzieć dokładnie, jak go rozwiązać. Jarek

dostaje szybką odpowiedź i nie musi spędzać wielu godzin, analizując ten problem samodzielnie. Każdy pracownik może wykorzystać doświadczenie całego zespołu w celu szybkiego rozwiązania problemów.

## Komunikacja w zespole

Codzienne spotkania zmuszają wszystkich do rozmów bez wyróżniania kogokolwiek. Spotykanie się każdego dnia na rozmowach oraz udzielanie sobie nawzajem pomocy wytwarza w zespole niespotykaną więź. Ile znacie zespołów, w których każdy pracownik rozmawia z każdym innym pracownikiem przynajmniej raz dziennie? Codzienne spotkania wiążą niejednorodną grupę programistów w prawdziwy zespół.

Typem programisty, który korzysta najwięcej z codziennych spotkań, jest człowiek nieśmiały, spędzający całe dnie na chowaniu się w swoim biurze. Pewna osoba, z którą pracowaliśmy (powiedzmy, że był to Ryszard), była modelowym przykładem takiej osobowości. Ryszard przychodził bezpośrednio do swojego pokoju, pracował do obiadu, który jadł sam, i wychodził na koniec dnia. Przez większość dni nie zamienił z nikim nawet słowa.

Gdy rozpoczęto organizowanie codziennych spotkań, wszyscy członkowie zespołu — nawet Ryszard — zaczęli się poznawać. Żarty opowiadano przed spotkaniami, w ich trakcie i po nich. Programiści pomagali sobie nawzajem w rozwiązywaniu problemów. Zawiązywały się przyjaźnie. Z tego, co ostatnio słyszeliśmy, Ryszard całkowicie wyszedł ze swojej skorupy i miał kontakty z prawie całym zespołem. Zapraszał innych na obiad, wpadał pogadać i utrzymywał kontakty na poziomie — wydawałoby się — wcześniej niemożliwym.

Widzieliśmy ten efekt u wielu ludzi na różnych etapach i zawsze z pozytywnymi wynikami.

## Szeroki punkt widzenia

Spotkania z innymi pracownikami oraz wiedza na temat tego, nad czym oni pracują, zapobiega „krótkowzroczności" w postrzeganiu funkcji projektu. Jeśli ta przypadłość dotyka pracowników, wydaje im się, że ich praca jest

jedyna i najważniejsza. Prowadzi to do iluzji wielkości i powoduje, że programiści przeceniają swoje miejsce w świecie. Często samo bycie w grupie, która dyskutuje nad różnymi projektami i kierunkami prac, może uleczyć taką osobę bez konieczności stosowania innych zabiegów.

Poza leczeniem „krótkowzroczności" cały zespół może odnieść korzyść z wiedzy o tym, czym zajmują się inni pracownicy. Jeśli jeden pracownik wie, że inny zamierza zmienić komponent, który on wykorzystuje, wówczas może zająć się innym obszarem kodu, tak aby sobie nawzajem nie przeszkadzali. Jeśli zmiana komponentu jest podobna do tej, którą wykonał już ktoś inny, wówczas możliwe jest udzielenie pomocy. Można dostrzec mnóstwo korzystnych efektów płynących z jawności informacji o tym, czym się każdy zajmuje.

## Inne możliwości

Istnieją pewne rozwiązania alternatywne wobec spotkań zespołów. Przyjrzyjmy się kilku z nich.

Menedżer lub kierownik techniczny może zaglądać od czasu do czasu do każdego pracownika. Daje to możliwość osobistej rozmowy z kierownikiem technicznym, co jest rzeczą pozytywną. Ogranicza to jednak kontakty pomiędzy pracownikami. Jest to również poważne marnotrawstwo czasu kierownika technicznego, który spędza dzień lub dwa każdego tygodnia na wizyty. Ponadto, zamiast chronić zespół przed przerwaniami, teraz on sam staje się ich powodem.

Jeszcze inna możliwość to dać wszystkim spokój przez kilka miesięcy, a następnie spróbować wszystko poskładać na koniec. Mimo że jest to często używany sposób, nie zalecamy go. Prowadzi on do wielu heroicznych wysiłków oraz licznych nadgodzin przy finalizowaniu projektu, ale nie przynosi w efekcie wielu działających produktów. Jeśli zespół pracuje w tego rodzaju próżni, nieporozumienia narastają. Jeśli wejdziesz na złą drogę, tracisz tygodnie lub miesiące, zanim sobie z tego zdasz sprawę. Pracowaliśmy w firmach, które działały w taki sposób, i za każdym razem obserwowaliśmy miesiące pracy programistów wyrzucane w błoto.

Bardzo popularne są spotkania cotygodniowe. Nasze zastrzeżenie do nich związane jest z ilością informacji, która musi być przekazywana, aby cotygodniowe spotkanie było naprawdę efektywne. Pracownicy wykonali w ciągu tygodnia taką pracę, że jakakolwiek sensowna wymiana informacji nie jest możliwa. Zamiast tego cotygodniowe spotkania stają się forum, na którym przekazywane są informacje oraz komunikaty od menedżera. Dobry menedżer może wykorzystać takie forum, aby skupić się na konkretnej kwestii lub problemie, ale nie może skupiać się na kwestii, o której istnieniu nie wie. Tygodniowe spotkania są lepsze niż kwartalne, ale ich efektywność jest znacznie mniejsza niż w przypadku spotkań codziennych.

## Jak wyglądają codzienne spotkania?

- 8:55 rano: pracownicy schodzą się do miejsca spotkania. Spotkanie jest częścią codziennych zajęć; prowadzone jest ono w tym samym pomieszczeniu o stałej porze. Czas przed spotkaniem wypełniony jest omawianiem takich istotnych kwestii jak ta, w której kawiarni jest najlepszy barista, albo nowości dotyczących rowerów górskich.
- 9:00 rano: kierownik techniczny Andrzej rozpoczyna spotkanie. Niekiedy znaczy to, że będzie musiał przerwać zażartą dyskusję dotyczącą nowych rowerów górskich. Następnie pracownik jest proszony o przedstawienie raportu o statusie. W tym przypadku rozpoczyna Krzysztof.

  Krzysztof pracował wczoraj nad dodaniem obsługi nowej bazy danych. Miał drobny problem ze składnią SQL-a, ale udało mu się go rozwiązać. Inny pracownik włącza się ze wskazówką, że również kilka innych poleceń SQL-a nie jest przenośnych. Teraz Krzysztof wie, że w jego zespole jest osoba, która również zajmowała się przenoszeniem programów w SQL-u, i może w każdej chwili zwrócić się do niej o radę.

- 9:03 rano: obok Krzysztofa siedzi Ania, więc ona odzywa się jako kolejna. Ania pisze prototyp aplikacji klienckiej, która prawdopodobnie będzie nowym interfejsem użytkownika. Spędziła dzień, programując ekran główny aplikacji. Przewiduje, że uda jej się dzisiaj skończyć dodawanie elementów GUI i że po obiedzie zacznie programować logikę ich działania.
- 9:04 rano: Michał rozbudowywał swój komputer i spędził dzień na instalacji oprogramowania. Kolejny szybki raport zakończony.
- 9:05 rano: Marek analizuje problem zgłoszony przez klienta. Wygląda na to, że część serwerowa produktu regularnie zawiesza się w poniedziałki rano, ale działa poprawnie w inne dni tygodnia. Jacek opowiada podobną historię, z którą zetknął się przy pracy z innym systemem. W jego przypadku serwer bazy danych był restartowany przez weekend i program zawieszał się, próbując uzyskać dostęp do bazy danych w poniedziałek rano. Jacek opowiada dalej całą historię. Po kilku minutach Andrzej przerywa i kończy opowieść. Jacek i Marek zgadzają się spotkać i przyjrzeć się razem problemowi.
- 9:10 rano: Andrzej przekazuje kilka komunikatów o wspólnym obiedzie w piątek oraz o nowym kliencie. Komunikaty są przekazywane pod koniec spotkania, ponieważ zawsze ktoś się spóźnia i nie usłyszałby ich, gdyby przekazywane były na początku.
- 9:15 rano: wszyscy są wolni i mogą wracać do pracy po krótkim piętnastominutowym spotkaniu.

## Od czego zacząć?

Jeśli nigdy nie przeprowadzałeś codziennych spotkań, oto kilka pomysłów, jak je zorganizować:

- Upewnij się, że każdy zna ich formułę (jakie pytania chcesz, żeby były stawiane).
- Każdy musi odpowiadać na te pytania. Nie ma wyjątków.

- Na początek bądź tolerancyjny, jeśli chodzi o ograniczenia czasowe. *Wiele* informacji jest przekazywanych na początku. Musisz pozwalać na swobodną komunikację.
- Wyznaczaj spotkanie codziennie o tej samej porze i w tym samym miejscu. Niech staną się one nawykiem, a nie udręką.
- Zamieszczaj tematy przedyskutowane na spotkaniu na stronie internetowej lub w *plogu*[6].

*plog*

- Wybierz osobę, od której zacznie się spotkanie, a następnie kolejne zgodnie z ruchem wskazówek zegara (lub w kierunku przeciwnym). Przypadkowe wybieranie jednego pracownika po drugim może ich niepotrzebnie deprymować.

## Nie rozpraszaj się

Codzienne spotkania będą wartościowe, jeśli każdy trzyma się tematu. Ważne jest, aby przekazywać informacje konkretnie. Nie mów, że skończyłeś w 70 procentach. Zamiast tego powiedz, że dodałeś ekran logowania i że chociaż jeszcze nie działa, to będzie działał jutro. Zatrzymaj się na chwilę, jeśli ktoś mówi, że ekran logowania zawiesza się. Poproś o bardziej szczegółowe opisanie problemu (np. „nie komunikuje się z narzędziem kontroli tożsamości"). Jeśli projekt dopiero się zaczyna, spotkanie prowadzi kierownik techniczny, ale z czasem obowiązek przewodniczenia powinien przechodzić rotacyjnie przez wszystkich pracowników. Wykorzystaj te spotkania do rozwijania liderów we własnej grupie.

Jeśli jakiś temat zaczyna żyć własnym życiem lub spotkanie zamienia się w debatę nad sposobem rozwiązania problemu, przewodniczący powinien szybko ją przerwać i polecić zaangażowanym osobom spotkać się w mniejszej grupie po głównym spotkaniu. Nie wszyscy muszą słuchać dyskusji dwóch pracowników nad problemem, który tylko ich dotyczy. Mogą oni później dać krótki przegląd rozwiązania, gdy już je znajdą.

---

[6] *Plog* jest to blog projektu. *Blog* to z kolei pamiętnik internetowy pozwalający na łatwą aktualizację. Użycie ploga do udostępniania informacji z codziennych spotkań ułatwia przekazywanie wiedzy pracownikom. Jest on podstawowym źródłem informacji dla pracowników nieobecnych, innych zespołów lub menedżerów.

## Jaś pyta...

### Nasze codzienne spotkania są zbyt długie. Co mamy zrobić?

Jeśli dopiero zaczynasz organizować codzienne spotkania, będą one trwały dosyć długo. Spotkania te służą wymianie informacji i w związku z tym trzeba wiele nadrobić. Celem jest, aby jedna osoba zajmowała jedną lub dwie minuty, ale są małe szanse, że osiągniesz ten cel w ciągu kilu dni, a nawet tygodni. Możesz również oczekiwać opóźnień w sytuacji, gdy w zespole pojawia się nowy pracownik. Codzienne spotkania są znakomitym forum pozwalającym mu się wdrożyć do pracy!

Jeśli po kilku tygodniach codzienne spotkania w dalszym ciągu trwają godzinę, wówczas należy ten problem rozwiązać. Prawdopodobnie pracownicy opisują swoje problemy oraz ich rozwiązania w sposób zbyt szczegółowy. Nie pozwalaj im opisywać szczegółów rozwiązania i staraj się uzyskiwać tylko podsumowania. Na przykład zamiast przedstawiania szczegółowej, niskopoziomowej analizy problemu, sposobu debugowania oraz ostatecznego rozwiązania, po prostu powiedz: „Mieliśmy problem z pamięcią podręczną, która się nie aktualizowała przy zmianie danych. Został on już naprawiony i wprowadzony do projektu". To wszystko, co musimy wiedzieć.

Możesz również poprosić pracowników, aby zapisali, co mają zamiar powiedzieć. Pomoże im to uporządkować myśli przed spotkaniem. W ten sposób unikną rozwlekłego raportu, który ciągnie się pięć minut.

Innym potencjalnym problemem jest zbyt duża liczba osób w zespole. Stwierdziliśmy, że w codziennych spotkaniach nie powinno brać udział więcej niż piętnaście osób. W przeciwnym razie należy podzielić spotkania na mniejsze grupy. Członkowie zespołu pracujący w tych samych obszarach powinni brać udział w tym samym spotkaniu. Postaraj się, aby przynajmniej jedna lub dwie osoby uczestniczyły w każdym spotkaniu, tak aby istotne informacje mogły być przekazywane.

Kilka poważnych metodologii programowania sugeruje, że czas spotkania można ograniczyć, każąc wszystkim stać. To działa, ale jeśli uczestnicy są zdyscyplinowani (lub gdy dyscyplinę potrafi narzucić przewodniczący), nie jest to konieczne.

## Postępujesz poprawnie, jeśli...

Jeśli organizujesz codzienne spotkania, to świetnie! Oto kilka sugestii:

- Czy spotkania są pożyteczne? Jeśli nikt w zespole niczego się nie uczy, to znaczy, że raporty mogą być zbyt zwięzłe. Jeśli w jakimś obszarze potrzebna jest większa szczegółowość, przesuń taki temat na poboczne spotkanie w mniejszej grupie. Jednak zalecenie dwóch minut jest tylko sugestią, a nie dogmatem. Może się okazać, że czasami wystarcza trzydzieści sekund, a czasami możesz potrzebować trzech minut.
- Czy spotkania odbywają się każdego dnia, w tym samym miejscu i o tej samej porze? Przeprowadzanie spotkań w tym samym miejscu i czasie powoduje, że łatwo o nich pamiętać. Spotkanie można przesunąć od czasu do czasu, ale unikaj częstych zmian.
- Jeśli przestałbyś przeprowadzać spotkania, czy pracownicy narzekaliby? Powinni! Zespół powinien polegać na codziennych spotkaniach, aby „być w temacie". Jeśli spotkań można zaprzestać, to znaczy, że nie miały one wartości. Zespół powinien polegać na codziennych spotkaniach jako na bezcennym zasobie.

## Sygnały ostrzegawcze

Codzienne spotkania są znakomitym narzędziem. Jednak jak każde narzędzie mogą być również szkodliwe, jeśli będą stosowane niepoprawnie. Oto kilka sygnałów ostrzegawczych świadczących o tym, że codzienne spotkania nie funkcjonują należycie:

- Każdy pracownik zajmuje dziesięć minut lub więcej.
- Jeden z pracowników *regularnie* zajmuje tyle samo czasu co wszyscy inni pracownicy razem wzięci.
- Uczestnicy przerywają sobie złośliwie. Żarty pomiędzy pracownikami są wskazane (i można do nich zachęcać), ale jeśli codzienne spotkania stają się okazją do wymiany złośliwości, przestają być efektywne.
- Spotkania regularnie zaczynają się (lub kończą) za późno.
- Spotkania stają się pozbawione treści. Programiści wygłaszają stwierdzenia w rodzaju „Skończyłem w 90 procentach" albo „Pracuję nad bazą danych".
- Pracownicy mówią chaotycznie lub zapominają przedstawiać zadania, które wykonali. Poproś ich na osobności, aby zapisywali, co zrobili, tak aby byli skupieni na spotkaniu i przedstawiali swoje sprawozdania zwięźle. Dobrze będzie również, jeśli będą mieli swój własny egzemplarz listy zadań, co pozwoli im być bardziej zorganizowanymi.

### Programowanie w parach

*Programowanie w parach* oznacza, że przy jednym komputerze pracuje dwóch programistów. Jeden pisze program, a drugi próbuje oceniać i patrzeć z dystansu. Jeden pracuje nad szczegółami programu i składni języka, a drugi próbuje zdecydować, czy dany algorytm jest właściwy do rozwiązania konkretnego problemu. Druga osoba zauważa problemy takie jak błędy w kodzie, literówki i błędne nazwy zmiennych. Od czasu do czasu programiści zamieniają się rolami.

Niektórym bardzo podoba się ta technika, inni nigdy nie potrafią się do niej przyzwyczaić. Stwierdziliśmy, że jest to bardzo przydatna technika, pod warunkiem że jest stosowana właściwie (z odpowiednimi osobami). Bardziej szczegółowe spojrzenie na tę intrygującą technikę znajduje się na stronie *http://www.pairprogramming.com*/.

# 13. Przeglądy kodu

Małe, częste przeglądy kodu powodują, że program jest przejrzysty, prosty i uporządkowany. Można uniknąć tradycyjnie nieprzyjemnych przeglądów kodu, które angażują dziesiątki programistów i wymagają dni przygotowań (znane także pod nazwą przeglądów MAD[7]). Stwierdziliśmy, że przeglądy kodu mogą być bezbolesne, jeśli będziemy się stosować do następujących reguł:

- Przeglądane będą tylko niewielkie fragmenty kodu.
- Przeglądu dokonuje jedna lub dwie osoby, nie więcej.
- Przeglądy dokonywane są bardzo często, nawet kilka razy dziennie.

*programowanie w parach*

Celem jest przyswojenie nawyku częstszych przeglądów kodu bez narażania się na potencjalny szok kulturowy (lub dodatkowe koszty) związany z *programowaniem w parach*. W wielu środowiskach nie ma gotowości do tego rodzaju współpracy; sam koszt cukierków miętowych mógłby pogrążyć dobrze prosperującą firmę. Zamiast tego staraj się dokonywać przeglądów kodu częściej i w mniejszych fragmentach.

Jeśli mija tydzień bez dokonania przeglądu kodu, to upłynęło wiele czasu, podczas którego w Twoim kodzie mogły się pojawić poważne błędy. Prawdopodobnie potrzebujesz spojrzenia z zewnątrz, jeśli pracowałeś nad poważnym problemem tak długo. Druga osoba niekoniecznie musi wiedzieć lepiej; samo wyjaśnienie komuś problemu często wystarcza do jego rozwiązania w następnej kolejności (czasem nazywa się to metodą *gumowej kaczki*[8]). Czekanie przez wiele dni na przegląd kodu (nawet jeśli jest to tymczasowa kontrola) skończy się prawdopodobnie długim i bolesnym przeżyciem... przeglądem MAD!

---

[7] Skrót od „Mighty Awful and Dreaded" (bardzo nieprzyjemny i wzbudzający strach). Skrót MAD można przetłumaczyć również jako „szalony" — *przyp. tłum.*

[8] Nazwa pochodzi stąd, iż słuchająca osoba nie musi nic wnosić do rozmowy. Wystarczy, że od czasu do czasu pokiwa głową. Jeśli nie możesz znaleźć takiej osoby, wystarczy Ci nawet gumowa kaczka.

Aby uniknąć przeglądów MAD, podziel swoje zadanie na najmniejsze możliwe kawałki i daj każdy z nich do niezależnego przeglądu, a następnie wprowadzaj je do repozytorium kodu źródłowego. Wówczas, jeśli istnieje problem z jakimś pojedynczym obszarem, jest on łatwo izolowany.

Programiści często są tak zaabsorbowani szczegółami określonego zadania, że nie patrzą na problem z dystansu, przez co mogą przeoczyć oczywiste usprawnienia. Jeśli zatrzymują się, aby wyjaśnić swój punkt widzenia oraz kod programu innej osobie, często zyskają świeżą, wartościową perspektywę. Czasem jesteśmy tak zajęci budowaniem drogi w lesie, że nie zauważamy, iż podążamy w złym kierunku!

Inna korzyść segmentacji kodu polega na możliwości łatwiejszego zrozumienia kodu przez osoby przeglądające. W szybko pracującej firmie programistycznej może istnieć potrzeba przeglądania kodu kilka razy dziennie. Dobrą zasadą jest, aby nie pracować dłużej niż dwa dni bez dokonania przeglądu kodu. Myśl o przeglądzie kodu jak o oddychaniu. Można oczywiście obyć się bez oddychania przez kilka minut, ale komu to potrzebne?

Idealnie będzie, gdy dla każdej dodanej funkcji (lub poprawionego błędu) zastosowany zostanie jeden przegląd kodu. Niesprawdzanie kodu, w czasie gdy zostanie dodanych siedem nowych funkcji i poprawionych zostanie czternaście błędów, jest gotową receptą na wzbudzający strach przegląd typu MAD (nie mówiąc o długim, przeciągającym się i nieskoordynowanym wysiłku).

Jeśli jesteś w sytuacji, gdy musisz przepisać zawiłą część swojego programu i nie potrafisz podzielić tego zadania na mniejsze części, wybierz jedną z osób dokonujących przeglądu i poleć jej dokonywanie częstych przeglądów kodu na bieżąco.

Będziesz pisał lepsze programy, jeśli będziesz wiedział, że ktoś inny je zobaczy. Spraw, że będziesz czuł się za nie odpowiedzialny. To zagadnienie nie dotyczy tylko programistów, taka jest po prostu ludzka natura. Przeglądy kodu zapewniają, że przynajmniej jedna osoba spojrzy na Twoją pracę. Mając tę odpowiedzialność, będziesz wiedział, że w swojej pracy nie możesz iść na skróty.

Wiele badań wykazuje efektywność przeglądu kodu w wykrywaniu błędów. W rzeczywistości jest to najważniejsza technika znajdywania błędów. Nie istnieje lepsza. Jeśli nie dokonywałeś regularnych przeglądów kodu, będziesz zdziwiony tym, co odkryjesz.

Zdarzało nam się widywać nazwy zmiennych w rodzaju PanHash dla tablicy hashującej. Jednak po jednym przeglądzie kodu programista zaczął używać bardziej sensownych nazw zmiennych, aby uniknąć dokuczania ze strony współpracowników. Podglądanie przez innych może być bolesne, ale będzie efektywne.

Gumowa kaczka (opisana wcześniej) jest bardzo efektywnym sposobem znajdywania i rozwiązywania problemów. Opisując swój program komuś, nagle zdasz sobie sprawę z tego, o czym zapomniałeś, rozpoznasz logikę, która jest niepoprawna, lub odnajdziesz niezgodności z innymi obszarami systemu. Chcemy, abyś „rozmawiał" z kaczką za każdym razem, gdy zatwierdzasz kod.

Rozmowa spowoduje również, że inni programiści *zauważą* błędy w Twoim programie. Druga para oczu patrząca na Twój program często znajduje problemy, które nawet nie przyszły Ci do głowy. Poznasz zupełnie inny punkt widzenia. Znajdowanie błędu w firmie programistycznej jest zawsze tańsze niż wówczas, gdy program jest już u odbiorcy. Stopa zwrotu z tej niewielkiej inwestycji jest naprawdę duża.

Przegląd kodu jest znakomitym narzędziem promującym dzielenie się wiedzą w firmie. Współpraca przy przeglądzie kodu spowoduje, że osoba przeglądająca ma przynajmniej ogólną ideę działania Twojego programu, a być może także jego głębsze rozumienie. Ma to ogromne znaczenie edukacyjne i pomaga również w utrzymaniu poprawności kodu.

Przeglądy stanowią znakomitą okazję dla doświadczonych programistów do przekazywania poprawnego stylu programowania i projektowania młodszym kolegom. Poza trywialnymi zaleceniami (na przykład dotyczącymi sposobu stawiania nawiasów) doświadczeni programiści mogą powiedzieć, dlaczego jedna struktura danych może w danej sytuacji pasować bardziej niż inna,

### Wzorce

*Wzorzec* odnosi się do praktyki dokumentowania i nazywania wspólnych problemów (i ich rozwiązań), które występują w projektach. Istnieje kilka powodów, aby zdobyć wiedzę na temat wzorców. Jednym jest wprowadzenie wśród programistów wspólnego słownictwa. Pracując ze sobą, programiści rozwijają wspólny zbiór pojęć, który pozwala im się komunikować szybko i jednoznacznie. Wzorce mogą przyspieszyć ten proces i pozwolić komunikować się jasno z kimś dopiero co spotkanym (pod warunkiem że osoba ta zna te same wzorce).

Innym powodem, dla którego warto znać wzorce, jest łatwość rozwiązywania problemów do tej pory niespotkanych. Poprzez czytanie i omawianie różnych wzorców uczysz się sposobów rozwiązywania wielu popularnych problemów. Pytanie brzmi, czy uda Ci się rozpoznać wzorce, kiedy się na nie natkniesz, a nie, czy uda Ci się spotkać z większością z nich*. Czy będziesz wiedział, jak elegancko rozwiązać problem opisywany przez wzorzec, czy też będziesz kilkakrotnie podchodził do rozwiązania, zanim znajdziesz właściwe?

*Wzorce projektowe. Elementy oprogramowania obiektowego wielokrotnego użytku* autorstwa Erica Gammy, Richarda Helmsa, Ralpha Johnsona i Johna Vlissidesa można polecić jako dobry punkt wyjścia do studiowania tematu.

* Artykuł „Humble Programmer" (Pokorny programista) autorstwa Edsgera W. Dijkstry przedstawia klasyczne spojrzenie na stan informatyki (i wzorców), które pozostaje w dalszym ciągu aktualne. Został on napisany w 1972 r. (patrz: *http://www.cs.utexas.edu/users/EWD/ewd03xx/EWD340.PDF*).

lub wskazać istnienie jakiegoś wzorca. Często osoba przeglądająca zauważy powtarzające się fragmenty kodu lub fragmenty programu, które można przenieść do wspólnych klas bazowych lub użytkowych. Dokonuje się refaktoringu kodu, *zanim* zostanie on zapisany w systemie zarządzania kodem źródłowym.

Przeglądy kodu ułatwiają także wspólne uczenie się, zarówno na małych obszarach kodu, jak i na ogólnych koncepcjach. Poza „stylem programowania" uczysz się także „programować ze stylem".

Oto kilka sugestii pomagających przeprowadzać przeglądy kodu.

Przegląd kodu musi angażować przynajmniej jednego dodatkowego programistę. W praktyce niemal zawsze wystarczy jeden dodatkowy programista, chyba że tworzysz coś na tyle interesującego, że inni pracownicy chcą się o tym dowiedzieć. Wówczas możesz śmiało zaangażować więcej programistów. Nie przesadzaj jednak (nie więcej niż trzech lub czterech) — zbyt wielu programistów spowolni przegląd.

Nie udostępniaj kodu publicznie bez przeglądu. Nie dodawaj swoich zmian do kodu źródłowego całego projektu, zanim nie zostanie dokonany przegląd. Częścią komentarza dostarczanego z kodem powinno być nazwisko osoby dokonującej przeglądu. Wówczas, jeśli pojawiają się pytania o przyczyny zmiany kodu, a Ciebie nie ma w pobliżu, jest inna osoba, która powinna być w stanie to wyjaśnić (przynajmniej na podstawowym poziomie).

### Refaktoring

Najlepszą definicję tego pojęcia podaje Martin Fowler:

„Refaktoring jest sformalizowaną procedurą zmiany wewnętrznej struktury istniejącego kodu bez zmiany jego zewnętrznego zachowania. Jego istotą jest szereg transformacji zachowujących dotychczasowe działanie programu. Pojedyncza transformacja (refaktoring) nie zmienia wiele, ale szereg transformacji może przynieść znaczące zmiany struktury. Dzięki temu, że pojedynczy refaktoring jest niewielki, istnieje mniejsza szansa, że spowoduje błędy. Utrzymuje się również pełne działanie systemu po każdym refaktoringu, co zmniejsza prawdopodobieństwo poważnej awarii podczas zmiany struktury"[*].

[*] Patrz: *http://www.refactoring.com/*.

Nigdy nie wykorzystuj przeglądu jako wymówki do niezatwierdzania kodu w projekcie. Jeśli w Twojej firmie jest system zarządzania kodem źródłowym, który przechowuje tylko kod produkcyjny, wówczas umieść swój kod w części prywatnej, dopóki nie będzie gotowy. Wymieniona prywatna część może być oddzielnym repozytorium kodu lub oddzielną instalacją systemu

zarządzania kodem źródłowym. Korzystaj z wszelkich narzędzi, aby przenieść kod ze swojego komputera, tak aby w przypadku jego awarii kod nie został utracony.

Osoby przeglądające kod mają prawo go odrzucić, jeśli uznają, że jest on nie do zaakceptowania. Jeśli przeglądasz kod innej osoby i nie jest on poprawnie skomentowany, algorytmy są nieefektywne lub powód jest jeszcze inny, nie bój się prosić o poprawki (jednak nie bądź wybredny — pamiętaj, że najczęściej istnieje wiele dobrych dróg osiągnięcia tego samego wyniku). Jako osoba przeglądająca kod masz za zadanie go poprawić, a nie bezmyślnie zatwierdzić. Jak mówi Eric S. Raymond, „Wiele par oczu sprawia, że błędy stają się płytkie".

Jeśli Twój program nie może być wyjaśniony na poziomie zrozumiałym dla osoby dokonującej przeglądu, wówczas musi być uproszczony. Jako przeglądający kod nie możesz podpisywać się pod czymkolwiek, czego nie rozumiesz lub z czym czujesz się nieswojo. Ostatecznie Twoje nazwisko jako recenzenta jest przypisane do tego programu. Podpisujesz się pod nim. Upewnij się, że jest on wart Twojego podpisu.

Wszystkie zmiany kodu muszą przechodzić pozytywnie istniejący zautomatyzowany zestaw testów. (Prowadzisz testy, prawda? Patrz: podrozdział 7. na stronie 63). Nie trać czasu swoich współpracowników, prosząc o przegląd kodu, jeśli nie przeprowadziłeś jeszcze testów. Jeśli wymagana jest aktualizacja istniejących testów, ich zmiany potraktuj jako część swojego zadania, a następnie przeprowadź przegląd. *Nowe* testy, które dodajesz, powinny również podlegać przeglądowi. Jako osoba dokonująca przeglądu odrzucaj zmiany kodu, jeśli uważasz, że konieczne są dalsze testy.

„Po pierwsze nie szkodzić"[9] to reguła, która ma zastosowanie również przy dokonywaniu przeglądu kodu. Zmiany kodu nigdy nie powinny unieruchamiać programu. Oczywiście reguła ta jest nieistotna, jeśli mamy dobry zestaw testów. Rzadko istnieje powód, aby zburzyć istniejącą funkcjonalność. Zamiast unieruchomić istniejące API, dodaj nowe, które będzie miało nowy argument (czy cokolwiek innego), którego potrzebujesz.

[9] Przysięga Hipokratesa.

Jeśli na przykład istniejące wywołanie procedury musi być zmienione, ustal plan, który zarysuje sposób eliminacji istniejącej implementacji. Nie usuwaj po prostu tej procedury swoim klientom (swoim współpracownikom lub innym zespołom w firmie); podejmij świadomą decyzję, czy zatrzymać starą procedurę, czy ją usunąć. Zaplanowane usuwanie jest także ważne — nie utrzymuj zdezaktualizowanych procedur przez lata.

Zmieniaj osoby, które przeglądają Twój kod, ale nie przywiązuj do tego nadmiernej wagi. Od czasu do czasu można mieć tę samą osobę przeglądającą kod, ale unikaj „kolesiostwa", gdy Ty zawsze przeglądasz kod kolegi, a on przegląda Twój. Ponadto nigdy nie twórz oddzielnego stanowiska dla pojedynczej (i przepracowanej) osoby przeglądającej kod wszystkim pracownikom. Obie sytuacje eliminują efekt przekazywania wiedzy, który starasz się wspierać.

Przeglądy kodu powinny być nieformalne. Zamiast planować spotkanie, raczej wpadnij do współpracownika i zapytaj, czy jest w tej chwili dobry moment na przegląd. Czasem można przeglądać kod, który jeszcze znajduje się w edytorze. Czasem wydrukujesz różnice w programach i zabierzesz je ze sobą. Sposób i miejsce nie są ważne.

Po wprowadzeniu procesu przeglądu kodu być może będziesz musiał wyznaczyć kilku starszych członków zespołu, aby obowiązkowo przeglądali każdy kod. Na początek jeden starszy pracownik będzie musiał obowiązkowo uczestniczyć w przeglądzie. Nie będziesz raczej potrzebował ich w tej roli dłużej niż kilka miesięcy. Kiedy pracownicy nauczą się podstaw, cały zespół będzie w stanie współdzielić odpowiedzialność. Jak mówi „Księga Przysłów", „Żelazo żelazem się ostrzy, a człowiek urabia charakter bliźniego"[10]. Celem jest, aby pracownicy wykonywali pracę razem i wzajemnie się od siebie uczyli. Angażuj swoich pracowników w proces „ostrzenia" tak szybko, jak to możliwe.

[10] Księga Przysłów 27:17, Biblia Tysiąclecia, Wydanie 3, Wydawnictwo Pallottinum, Poznań – Warszawa 1980 — *przyp. tłum.*

Pracowaliśmy w pewnej firmie, której funkcjonowanie znakomicie ilustruje, jak przeglądy kodu mogą być zastosowane do skorzystania z doświadczenia starszych programistów. Mieliśmy trzech bardzo doświadczonych pracowników i pięciu mniej doświadczonych, ale nie nowicjuszy, którzy jednak mieli czasami nieco dziwne pomysły na rozwiązywanie swoich zadań. Aby zabezpieczyć program przed awariami i podnieść kwalifikacje młodszych pracowników na wyższy poziom, wszystkie przeglądy kodu angażowały przynajmniej jednego doświadczonego pracownika. Dzięki temu bardziej doświadczeni pracownicy znajdowali problemy przed ich wprowadzeniem do produktu, jednocześnie ucząc młodszych. Uświadamiało to jednocześnie starszym pracownikom nieporozumienia i prawdziwe problemy, przed którymi stawali młodsi.

Te przeglądy stanowiły wielką pomoc dla zespołu. Często zauważaliśmy powtarzający się kod, który przenosiliśmy do klas użytkowych. Wychwytywany i usuwany był kod, który nie miał nic wspólnego z wyznaczonym zadaniem, oraz odrzucane były fragmenty pozbawione komentarzy. W miarę upływu czasu zaszła słabo widoczna, ale bardzo ważna zmiana.

Każdy z młodszych programistów zaczynał nabierać dobrych nawyków. Zaczynali porządkować swój kod przed przeglądem, dodając zrozumiałe nazwy zmiennych oraz komentarze, zanim ich o to poproszono. Długie, skomplikowane procedury stawały się krótkie i zrozumiałe.

Co więcej, nawyki nabyte w trakcie przeglądów kodu utrwaliły się. Po około trzech miesiącach zmieniliśmy politykę przeprowadzania przeglądów kodu tak, że każdy pracownik mógł ich dokonywać.

Jeśli jeden lub dwóch Twoich programistów regularnie nie zauważa istotnych kwestii w trakcie przeglądów, powinieneś kontrolować ich pracę za pomocą powiadomień o zmianie kodu (patrz: podrozdział 14. na stronie 128). Monitorowanie powiadomień o zmianie kodu pozwala w prosty, nieinwazyjny sposób mieć pod kontrolą każdego pracownika bez siedzenia w jego biurze i zaglądania mu przez ramię.

Czasami będziesz tak pochłonięty problemem, że przerwa na wykonanie przeglądu kodu całkowicie wybiłaby Cię z rytmu. Próba powrotu do własnego problemu kosztowałaby Cię mnóstwo czasu.

(Czy pamiętasz dyskusję o przerwaniach? Patrz: podrozdział 11. na stronie 93). Jeśli ktoś przychodzi i prosi o przegląd kodu (zresztą również o cokolwiek innego), kiedy jesteś pochłonięty problemem, powiedz mu, że jesteś zajęty teraz, i poproś, żeby przyszedł później. Z drugiej strony, jeśli to Ty szukasz kogoś do przejrzenia kodu i ktoś mówi, że jest zajęty, daj mu spokój i zaczekaj na lepszy moment lub idź do kogoś innego.

Większość pracy związanej z programowaniem to praca umysłowa — jesteś pochłonięty problemem, aż będzie on rozwiązany. Nie jest niegrzeczne poprosić kogoś, aby przyszedł później, jeśli jesteś w sytuacji, która wymaga koncentracji. W niektórych firmach jest to naturalne, w innych całkiem obce. Zawsze powinna być zrozumiana prośba, aby ktoś przyszedł za pół godziny albo po obiedzie.

### Wirtualne przeglądy kodu

Z czasem nauczysz się, na co zwracają szczególną uwagę poszczególne osoby przeglądające kod. Na przykład kiedyś Jarek napisał skomplikowany fragment programu, który — ku jego satysfakcji — działał. Następnie przeprowadził „wirtualny przegląd", próbując dostrzec, na co dwóch z jego najbardziej doświadczonych pracowników może zwrócić uwagę. Po wprowadzeniu zmian, co do których spodziewał się, że zostaną mu zaproponowane, pozwolił im przeprowadzić rzeczywisty przegląd kodu. Byli zachwyceni! Przeglądali razem już tak dużo programów, że był w stanie przeanalizować kod z ich punktu widzenia. Jarek nauczył się, co dwaj programiści (z wieloletnim doświadczeniem) sobie cenili, i był w stanie wykorzystać to doświadczenie, aby poprawić swoją pracę. Taki jest więc cel przeglądów kodu — nie tylko opracowujesz dobre programy, ale również wychowujesz dobrych programistów.

**▸ WSKAZÓWKA 17**

Można mówić „później”

Kierownictwo firmy powinno *wymagać* przeglądu kodu. Jeśli nie ma zalecenia kierownictwa, nikt z firmy nie ma oficjalnej motywacji, aby w tym uczestniczyć. Innymi słowy, jeśli nikt nie dostał polecenia, aby Ci pomóc, prawdopodobnie nie będzie miał dla Ciebie czasu, zwłaszcza jeśli terminy są napięte.

Jeśli w firmie nie ma polityki obowiązkowych przeglądów kodu, także wtedy możesz prosić swoich kolegów o przegląd. Nie cały zespół odniesie korzyść, ale Twój własny kod będzie lepszy. Osoby dokonujące przeglądu Twojego kodu z czasem nauczą się, jakie korzyści z tego płyną.

Nie czekaj zbyt długo na konkretną osobę, aż będzie miała czas, by dokonać przeglądu Twojego kodu; przejdź się po firmie i znajdź kogoś, kto nie jest pochłonięty problemem. Idź na drugi koniec budynku, jeśli musisz, ale znajdź kogoś. Jeśli znajdziesz kogoś, kto nigdy nie wykonywał przeglądu kodu dla Ciebie, będzie to dla niego świetna okazja, aby dowiedzieć się, co robisz.

Te szybkie przeglądy kodu promują wzajemną edukację bez dodatkowego kosztu sformalizowanych programów. Zmieniając programistów, którzy dokonują przeglądu Twojego kodu, osiągasz korzyść wynikającą z korzystania z doświadczenia i wiedzy wielu różnych osób. Każdy przeglądający kod wskaże różne sposoby rozwiązania tego samego problemu. Niektóre lepsze, niektóre gorsze, ale zawsze różne.

**▸ WSKAZÓWKA 18**

Zawsze przeglądaj cały kod

Celem jest kreatywne myślenie przy jednoczesnej poprawie jakości programu. Naucz się spoglądać na swoje problemy z różnych punktów widzenia. Krótkie przeglądy kodu staną się wkrótce Twoją drugą naturą

i — podobnie jak z kuchenkami mikrofalowymi — będziesz się zastanawiał, jak kiedyś bez nich żyłeś. Praktyczna dyskusja nad analizą algorytmu lub ograniczeniami zasobów jest lekcją, która jest zapamiętywana, ponieważ wiąże się z praktycznym i natychmiastowym zastosowaniem.

Zamiast uczyć się z książek, będziesz siedział przy biurku różnych fachowców (niektórych świetnych, innych przeciętnych) — będziesz uczył się trochę od każdego z nich, dodawał ich triki do własnych, aż pewnego dnia sam zostaniesz mistrzem.

## Od czego zacząć?

Przeglądy kodu to wspaniałe narzędzie! Kiedy staną się one nawykiem, będziesz się zastanawiał, jak kiedykolwiek bez nich pisałeś przyzwoite programy. Skorzystaj na początek z poniższych wskazówek:

- Upewnij się, że każdy wie, jakiego rodzaju przegląd kodu masz na myśli. Dokonuj przeglądu często, na małych fragmentach kodu. Nie czekaj tygodniami, aż przybędzie kilkaset lub kilka tysięcy wierszy zmian kodu. Niech Twój zespół będzie wolny od szalonych przeglądów (MAD)!
- Niech jeden z doświadczonych programistów uczestniczy we wszystkich przeglądach kodu przez pierwsze kilka tygodni lub miesięcy. Jest to znakomity sposób wymiany wiedzy, a ponadto umieszcza przeglądy na solidnym fundamencie.
- Przeglądy kodu powinny być niewielkie. Lepiej jest przejrzeć zbyt mały fragment programu niż zbyt duży. Przeprowadzanie dwóch przeglądów jest lepsze niż jednego dużego.
- Wprowadź system powiadamiania o zmianach kodu (patrz: podrozdział 14. na stronie 128). Stanowi on znakomite uzupełnienie przeglądów kodu, a ponadto pomaga przypomnieć pracownikom o tym, kto zapomniał poprosić o przegląd.
- Upewnij się, że masz akceptację zarządu, zanim zaczniesz wymagać uczestnictwa w przeglądach od wszystkich pracowników.

## Postępujesz prawidłowo, jeśli...

- Czy przeglądy kodu uzyskują automatyczną akceptację? To się nie powinno zdarzać, chyba że wszyscy pracownicy w zespole są doskonali.
- Czy każdy przegląd kodu wywołuje poważne zmiany w analizowanym programie? Jeśli tak, oznacza to, że w jakimś miejscu istnieje problem: albo z programistą, albo z przeglądającym kod, albo z kierownikiem technicznym (który dał zalecenia dla programisty i przeglądającego odnośnie charakteru przeglądu).
- Czy przeglądy kodu wykonywane są często? Jeśli czas pomiędzy nimi wynosi kilka tygodni, znaczy to, że czekasz zbyt długo.
- Czy osoby wykonujące przeglądy kodu zmieniają się?
- Czy zyskujesz nową wiedzę poprzez przeglądy kodu? Jeśli nie, powinieneś zacząć zadawać więcej pytań w trakcie ich trwania.

## Sygnały ostrzegawcze

- Przeglądy kodu są sporadyczne.
- Większość przeglądów kodu jest nieprzyjemna.
- Pracownicy unikają kończenia swoich fragmentów programu, ponieważ nie chcą ich przeglądu.
- Pracownicy, którzy przeprowadzili przegląd kodu, nie potrafią wyjaśnić jego funkcji lub dlaczego został napisany.
- Młodsi pracownicy przeglądają kod tylko innych młodszych pracowników.
- Starsi pracownicy przeglądają kod tylko innych starszych pracowników.
- Jeden pracownik jest ulubioną osobą przeglądającą kod dla wszystkich pracowników.

**Radiator informacji**

Według Alistaira Cockburna:

„Radiator informacji wyświetla informacje w miejscu, gdzie przechodzący mogą go zobaczyć. Nie muszą oni zadawać pytań; informacja sama do nich dociera, gdy go mijają".

„Dobry radiator informacji ma dwie cechy. Pierwsza z nich oznacza, że informacja zmienia się w czasie. Dzięki temu wart jest poświęcenia chwili, aby na niego spojrzeć. Drugą cechą jest to, że wymaga bardzo niewiele energii, aby wyświetlić informację".

Patrz: [Coc01].

# 14. Wysyłanie powiadomień o zmianie kodu

Po dokonaniu zmiany kodu system automatycznej kompilacji i konsolidacji zauważa ją i dokonuje rekompilacji projektu (patrz: podrozdział 4. na stronie 49). Następnym krokiem jest udostępnienie tej informacji tak, aby każdy pracownik wiedział, jaka zmiana została dokonana.

System powiadamiania o zmianie przekazuje tę informację do całej firmy, a nie tylko do bezpośrednich współpracowników. Efekt tego rodzaju przekazywania wiedzy może być niezwykły.

Udostępniasz informację podobnie jak w przypadku radiatorów informacji Alistaira Cockburna. Zespół może ją wykorzystać lub zignorować, ale zawsze ma do niej dostęp. Omawiana zasada nie wymaga od Ciebie sięgania po tę informację; informacja sama wpływa na Twoje biurko.

Za każdym razem, gdy wprowadza się ten standard postępowania, znaczny odsetek pracowników sprzeciwia się mu, zanim go nie wypróbuje.

Po około miesiącu najgorsi sceptycy zawsze przychodzą i przepraszają, mówiąc jak użyteczne stało się dla nich to narzędzie. Technika ta niezmiennie wywołuje największy opór, ale po krótkim czasie każdy przyzwyczaja się do dostępności powiadomień. Szybko staje się ona niezbędnym zasobem.

Typowe powiadomienie o zmianie kodu może wyglądać następująco:

```
W pakiecie bazy danych dodano createRecord() do TdDataFile
Index: com/tde/db/TdDataFile.java
===================================================================
RCS file: c:\apps\cvs\cvsroot/WeissDB/com/tde/db/TdDataFile.java,v
retrieving revision 1.3
diff -r1.3 TdDataFile.java
381a382,401
> /**
> * Zwraca nową instancję typu rekordowego przechowywanego w tej tabeli.
> * @return TdRecord - Nowa instancja rekordu.
> * @exception ClassNotFoundException - Generowany jest wyjątek, jeśli
> * nazwa klasy nie może być znaleziona.
> * @exception InstantiationException - Generowany jest wyjątek, jeśli
> * nazwa klasy nie może być utworzona.
> * @exception IllegalAccessException - Generowany jest wyjątek, jeśli
> * klasa reprezentująca zachowaną nazwę klasy ma błędny zasięg lub
> * nie ma zerowego parametru ctor.
> */
> public TdRecord createRecord() throws ClassNotFoundException,
> InstantiationException,
> IllegalAccessException {
> // Tworzenie nowej instancji rekordu
> return (TdRecord) (Class.forName(getRecordName()).newInstance());
> }
>
>
```

Istnieją dwa sposoby implementacji tego typu systemu. Preferowaną metodą jest automatyczne wysyłanie informacji o zmianach e-mailem wszystkim pracownikom. Większość systemów automatycznej kompilacji i konsolidacji będzie wysyłało zmiany Tobie (i zwykle opublikują je na stronie internetowej, a także poprzez kanał RSS). Drugi sposób polega na ręcznej implementacji tego systemu. Każdy pracownik musi zidentyfikować różnice w swoim kodzie w stosunku do poprzedniej wersji i *wysłać je e-mailem* pozostałym pracownikom.

Jakkolwiek zostanie to zaimplementowane, powiadomienia o zmianach powinny być wysyłane do pracowników za każdym razem, gdy kod źródłowy zostaje zapisywany do systemu zarządzania kodem źródłowym. E-maile zawierające powiadomienia powinny uwzględniać następujące informacje:

- Nazwisko przeglądającego kod.
- Cel zmiany kodu lub jego uzupełnienia (na przykład jaki błąd został poprawiony lub jaka nowa funkcja została dodana).

- Różnice pomiędzy nowym a starym kodem (każdy poważny system zarządzania kodem źródłowym wygeneruje dla Ciebie odpowiedni raport). Jeśli kompletnie przepisałeś fragment kodu na tyle duży, że wskazywanie różnic byłoby bezcelowe, wówczas tylko wstaw nowy kod. To samo odnosi się do nowych plików.

## Nieoczekiwane korzyści

Kilka lat temu dokonywaliśmy refaktoringu dosyć skomplikowanego i wolno działającego algorytmu, próbując go przyspieszyć. Po skończeniu przesłaliśmy innym pracownikom zwyczajowe powiadomienie o zmianie kodu. Godzinę później pewien programista przyszedł do nas z całkowitą zmianą algorytmu. Jego zmiany powodowały, że algorytm działał o rząd wielkości szybciej.

Programista ten miał doktorat w dziedzinie obejmującej zaawansowaną analizę algorytmów. Wszyscy o tym wiedzieli, ale ponieważ ostatnie miesiące były dosyć gorączkowe, nikomu nie przyszło do głowy, aby poprosić go o przejrzenie zmian. Jednak kiedy otrzymał e-maila i przeczytał jego nagłówek, zwrócił na niego uwagę. Ze swoją wiedzą natychmiast zauważył „oczywistą" poprawkę całego algorytmu, o której nikt inny nie pomyślał. To opowiadanie jest klasycznym przykładem niespodziewanych korzyści osiąganych z powiadomień o zmianie kodu.

## Informacja dostępna dla wszystkich

Powiadomienia o zmianie kodu stanowią prostą metodę rozwijania odpowiedzialności wśród pracowników. Pomagają one znaleźć niezdyscyplinowanych programistów, którzy nie wykonują przeglądów kodu lub dodają kod niezwiązany bezpośrednio z błędem lub nową funkcją wymienioną na liście zadań.

Bardzo szybko zauważysz najbardziej aktywnych programistów i przeglądających programy. Jeśli ktoś nie udostępnia żadnego kodu przez tydzień, kierownik techniczny może pójść złożyć wizytę, aby sprawdzić, czy dana

## Podpisuj się pod swoją pracą

Seria *Pragmatyczny programista* przypomina nam, że zawsze powinniśmy pracować w taki sposób, aby czuć się dumni ze swoich osiągnięć. Niezależnie od tego, czy Twoja praca polega na pisaniu programów czy budowie katedry, zawsze powinieneś pracować w taki sposób, jakby wszystko, czego dotkniesz, miało być sprawdzone przez Twoich współpracowników i klientów w bardzo jasnym świetle. Czy gdybyś wiedział, że drewniany panel, który produkujesz, będzie położony przy głównym wejściu do katedry, a nie w tylnej części strychu, czy pracowałbyś nad nim inaczej?

Podpisanie się pod swoją pracą *nie* oznacza, że masz wyłączne prawa do fragmentu programu i nikt go nie może modyfikować. Oznacza to, że pracowałeś nad danym programem i będziesz odpowiadał za swoje modyfikacje. Jeśli ktoś będzie chciał zadać pytanie o program, Twoje nazwisko będzie jako pierwsze na liście. Nazwiska pracowników, którzy przeglądali Twój program, będą widniały dalej. Jeśli ktoś znajdzie błąd w Twoim programie, będzie wiedział, kto go modyfikował i kto może dany błąd poprawić.

Nabieranie umiejętności w pracy polegać będzie na zostaniu ekspertem w konkretnych fragmentach kodu źródłowego projektu — nie ma w tym niczego złego. Specjalizacji podlegają lekarze i większość innych zawodów. Możesz pójść do swojego lekarza rodzinnego na coroczną kontrolę zdrowia, ale jeśli znajdzie on jakieś podejrzane objawy, szybko pójdziesz do specjalisty. Oczekiwanie, aby każdy programista w zespole nauczył się każdego fragmentu kodu, nie jest realistyczne w spotykanych na co dzień projektach. Podpisując się pod swoją pracą, przekazujesz do publicznej wiadomości, że to jest fragment programu, który znasz, i możesz nad nim pracować. Zmuszanie programistów do zmiany swoich ról nie oznacza, że każdy stanie się ekspertem w każdej dziedzinie. Oznacza to, że nikt nie stanie się ekspertem w *żadnej* dziedzinie. Mimo że mogą istnieć przypadki, w których utrzymywanie wszystkich na tym samym poziomie będzie wskazane, na ogół wolimy mieć wokół siebie specjalistów — zarówno przy programowaniu, jak i w medycynie.

osoba nie utknęła w miejscu lub nie zeszła z wytyczonego szlaku. Szef pewnej firmy poprosił nas o dodanie go do listy powiadomień, tak aby mógł z bliska przyglądać się projektowi. Orientacja kierownictwa w sprawach projektowych jest trudna do osiągnięcia w przypadku większości projektów programistycznych i to jest dobra droga na udostępnianie informacji.

## Możesz je ignorować

Możesz postanowić ignorować powiadomienia o zmianach kodu. W końcu jednak prawdopodobnie zauważysz, że spoglądasz na nie od czasu do czasu. Zanim się zorientujesz, stwierdzisz, że jesteś od nich uzależniony i będziesz z nich korzystał codziennie.

Kiedy nasz redaktor, Andy Hunt, przeglądał ten tekst, zaznaczył ten paragraf do wyjaśnienia. (*To namawianie do skasowania tych e-maili — red.*). Przez przypadek (i zanim mieliśmy czas porozmawiać z nim o tym paragrafie), odwiedzając klienta, zapoznał się tą techniką i stwierdził, że sprawdza się ona bardzo dobrze. Widząc jej praktyczne korzyści, pozwolił nam pozostawić ten paragraf.

## Od czego zacząć?

Istnieje wiele sposobów wprowadzenia powiadomień o zmianie kodu. Korzystaliśmy zarówno z systemu ręcznego, jak i automatycznego. W przypadku systemu ręcznego e-maile pisane są własnoręcznie (różnice w kodach wklejane są ręcznie) i następnie wysyłane do odbiorców. W przypadku systemu automatycznego program w ramach systemu zarządzania kodem źródłowym generuje powiadomienia i wysyła je e-mailem.

Oba podejścia są dopuszczalne, ale lepszy jest system automatyczny. Jego instalacja w niektórych środowiskach może być jednak utrudniona. Jeśli tak jest, skorzystaj z systemu ręcznego.

Upewnij się, że Twoi pracownicy zostaną poinformowani o powiadomieniach, zanim zaczną one do nich trafiać.

### Postępujesz prawidłowo, jeśli...

Powiadomienia muszą być regularne i wiarygodne.

Nie wysyłasz pliku zawierającego różnice o objętości pięciu megabajtów.

### Sygnały ostrzegawcze

Jedynym problemem, który może pojawić się w tym obszarze, jest wiarygodność. Jeśli pracownicy nie będą mieli pewności, że e-maile będą wysyłane po zmianach kodu, nie będą na nich polegali. Problem ten łatwiej się rozwiązuje w przypadku systemu automatycznego, ale w każdej sytuacji należy się temu zagadnieniu przyjrzeć.

## 15. Podsumowanie

W odróżnieniu od skomplikowanych i rozbudowanych metodologii podane zalecenia naprawdę *możesz* zastosować. Możesz zacząć od dzisiaj i niemal natychmiast zobaczyć pierwsze korzyści. Z czasem zauważysz ich oczywiście więcej i czym więcej zainwestujesz w metody swojej pracy, tym więcej ich odniesiesz. Nie istnieje złoty środek, ale procedury, które przedstawiliśmy, zapobiegają dużej liczbie popularnych katastrof. Nie będą przy tym szczególnie uciążliwe. Wprowadzenie tych procedur sprawi, że będziesz wyglądał jak bóg programowania, ale tak naprawdę będziesz tylko korzystał z doświadczeń innych. „Staniesz na barkach olbrzymów", jak brzmi powiedzenie.

Oto podsumowanie technik, które omówiliśmy w bieżącym rozdziale. Skopiuj je, powieś na ścianie i *przestrzegaj ich*. Wkrótce będziesz się zastanawiał, jak sobie bez nich radziłeś.

- Lista zadań powinna:
  - Być ogólnie dostępna.
  - Mieć przypisane priorytety.
  - Mieć prognozy czasu wykonania.
  - Być dynamiczna.

- Kierownik techniczny:
  - Zarządza listą funkcji projektu.
  - Monitoruje bieżące zadania oraz status programistów.
  - Pomaga przypisać każdej funkcji priorytet.
  - Izoluje pracowników od zakłóceń z zewnątrz.
- Codzienne spotkania powinny:
  - Być krótkie.
  - Być zwięzłe.
  - Sygnalizować problemy, ale ich nie rozwiązywać.
- Przeglądy kodu:
  - Dotyczą małych fragmentów programu.
  - Powinny być wykonywane przez jedną lub dwie osoby.
  - Powinny być wykonywane często.
  - Powinny poprzedzać opublikowanie kodu.
- Powiadomienia o zmianie kodu:
  - Wysyłane są e-mailem.
  - Zawierają nazwisko osoby przeglądającej kod.
  - Wymieniają cel zmiany lub uzupełnienia kodu.
  - Zawierają plik z różnicami pomiędzy wersjami programu.