Jak robić dobry code review

Robienie dobrych code review to umiejętność, którą warto opanować. W artykule omówimy, jakie są zalety i wady tego procesu i w jakich projektach warto go stosować. Zastanowimy się, jakie podejście warto przyjąć, robiąc review kodu, w jaki sposób najlepiej to robić, na jakich aspektach kodu możemy się skupić i wreszcie – w jakiej formie pisać komentarze z naszymi uwagami do kodu, aby służyły one dla dobra projektu, a cały ten proces był okazją do owocnej komunikacji między członkami zespołu, a nie źródłem konfliktów.

Dostałeś/aś do zrobienia swój pierwszy code review i nie wiesz, jak się za to zabrać? Na co zwracać uwagę, przeglądając czyjś kod? W jakiej formie pisać komentarze? A może kod, który ty piszesz, będzie podlegał review innych osób i nie wiesz, czego się spodziewać ani jak zareagować? Jeżeli tak, to ten artykuł jest dla ciebie! Nawet jednak jeżeli jesteś już doświadczonym programistą, znajdziesz tu zestaw porad, z którymi być może się zgodzisz, a być może nie, ale z pewnością dadzą one do myślenia.

Mówimy tu oczywiście o procesie w wytwarzaniu oprogramowania, w którym jeden programista dostaje do przejrzenia kod innego programisty i ma za zadanie zgłosić swoje komentarze oraz zatwierdzić go, jeżeli wszystko jego zdaniem wygląda OK. Można by po polsku określić go na przykład jako „przeglądanie kodu", ale ja pozwolę sobie używać angielskiego terminu „**code review**" ze względu na to, że jest on powszechnie używany.

To praktyka może przybierać różne formy w różnych organizacjach i zespołach. Może to być nieformalne zawołanie kolegi z biurka obok, aby spojrzał przez ramię na kod wyświetlony na ekranie i powiedział, co o nim myśli. Może też być to, z drugiej strony, w pełni sformalizowany proces, który używa specjalnego narzędzia w ramach systemu kontroli wersji (np. „pull request" na GitHubie albo Perforce Swarm) i wymaga od osoby przeglądającej zatwierdzenia, aby odblokowana została możliwość zrobienia merge czy też submit.

Będziemy tu omawiali code review jako proces w ramach pracy w danej organizacji, w danym zespole, nad danym projektem. Wszystko to samo może jednak dotyczyć także prowadzenia projektów **open source**, gdzie uwagi zgłaszane przez różnych użytkowników np. za pomocą „issues" i „pull requests" na GitHubie też stanowią tak jakby code review danego (publicznie dostępnego) kodu.

KIEDY I PO CO ROBIĆ CODE REVIEW?

O tym, czy w ogóle warto wprowadzić praktykę code review w danym projekcie, należy zdecydować indywidualnie, biorąc pod uwagę charakter projektu oraz znając wady i zalety tej praktyki. Spośród **zalet** pierwsze do głowy z pewnością przychodzi podniesienie jakości kodu. Dodatkowa para oczu skierowana na każdy nowy czy modyfikowany fragment kodu to okazja, aby ocenić go z innej perspektywy, a więc być może znaleźć i zgłosić ewentualne błędy, zanim trafią do repozytorium. Mogą to być błędy logiczne, nieprawidłowo obsłużone „przypadki skrajne" (np. kiedy dane wejściowe są puste), wycieki pamięci, problemy z wydajnością czy nawet luki bezpieczeństwa.

Istnieją też jednak inne, mniej oczywiste zalety wykonywania code review. Wprowadzenie tej praktyki sprawia, że każdy fragment kodu widziały i znają w mniejszym lub większym stopniu co najmniej dwie osoby. Prowadzi to do pożądanego w każdym zespole rozpowszechnienia wiedzy, zamiast zamykać każdego inżyniera w „silosie" tematu, którym się zajmuje. Dzięki temu być może ktoś inny będzie mógł w przyszłości – w razie potrzeby – łatwiej wrócić do pracy z danym kodem. Innymi słowy, zwiększa się tzw. „bus factor". Wreszcie, sama świadomość, że kolega czy koleżanka z zespołu przeczyta nasz kod, a nie wyłącznie kompilator, może nas zmotywować do większej staranności w jego pisaniu tak, aby nie tylko działał poprawnie, ale i był dobrze przemyślany i czytelny.

Możemy też oczywiście wymienić **wady** wykonywania code review. Pierwszą i najbardziej oczywistą jest cenny czas programistów, jaki należy na to poświęcić. Każdy czas pracy spędzony na przeglądaniu i komentowaniu cudzego kodu to czas, którego nie poświęcamy na pisanie albo debugowanie swojego kodu. Z punktu widzenia osób zarządzających zasobami i czasem programistów może to się wydawać wręcz jego marnowaniem, a przynajmniej czynnikiem obniżającym produktywność pracy. Nie bez znaczenia jest też konieczność oderwania się od swoich zadań i skierowania uwagi na inny fragment programu, co bywa obciążające mentalnie.

Argumentem często podnoszonym przez przeciwników włączania code review do procesu jest stwierdzenie, że błędy w kodzie powinny raczej być znajdowane przez dobrze napisane i regularnie wykonywane testy – jednostkowe, integracyjne itd., ewentualnie ręczne testowanie programu przez dział QA, a także inne automatyczne narzędzia, jak statyczna analiza kodu. Jest to oczywiście prawda, tak jak prawdą jest, że człowiek czytający kod nie „przetestuje w głowie" tylu możliwych przypadków, ile sprawdzą dobre testy.

Tym niemniej jedno nie wyklucza drugiego. Można powiedzieć wręcz, że testy automatyczne i code review świetnie się uzupełniają. Reviewer czy reviewerka może zwrócić uwagę na jakiś przypadek, który nie jest pokryty testami. Poza tym testy w ogóle nie sprawdzają istotnych aspektów kodu, takich jak jego struktura, czytelność, trafność nazw zmiennych i funkcji, komentarze czy dołączona dokumentacja, które mogą być kluczowe w przyszłym utrzymaniu kodu.

Jeszcze inną potencjalną wadą code review jest wydłużenie procesu. Konieczność zatwierdzenia każdej zmiany w kodzie, podobnie jak konieczność przejścia testów automatycznych, może spowodować, że każda, nawet najmniejsza zmiana trafia do repozytorium co najmniej po jednym lub kilku dniach od napisania. Taka sytuacja może być akceptowalna zależnie od charakteru projektu i tego, jak szybko na-

leży iterować jego kod. Na efektywność procesu review ma też oczywiście wpływ organizacja tego procesu. Jeżeli wystarczy, że dowolny inny członek zespołu zatwierdzi zmianę, wszystko to może się dziać szybciej, niż kiedy każde review zatwierdza jeden lider zespołu, który w dodatku jest bardzo zapracowany lub akurat poszedł na urlop.

Podsumowując tę część, można powiedzieć, że to, **czy warto robić code review**, zależy od charakteru projektu. Jeżeli do napisania jest prototyp, jakieś narzędzie do użytku wewnętrznego czy też rozwiązanie tymczasowe, to być może nie warto tego robić, bo jakość kodu nie musi być jak najwyższa. Wówczas bardziej liczy się szybkość jego tworzenia, modyfikowania oraz zwinność całego procesu.

Z drugiej strony, jeżeli projekt dotyczy rozwoju i utrzymania systemu, który trafia do klientów, być może stanowi główny produkt naszej firmy, bądź też jest platformą, na bazie której działa inne oprogramowanie (np. kompilator, system operacyjny), to jakość kodu powinna być priorytetem. Zawsze jednak o tę jakość w jakimś co najmniej minimalnym stopniu warto dbać – nawet jeżeli nie wymuszają tego rozbudowane procesy. Wiemy w końcu, że „prowizorki bywają najtrwalsze". Podsumowaniem tego rozdziału może być Tabela 1.

NIEKONIECZNIE	TAK
Projekt mniej ważny, np. narzędzie pomocnicze	Projekt ważny, np. główny produkt naszej firmy
Liczy się czas – szybkie iterowanie	Liczy się jakość – jak najmniej błędów
Projekt tymczasowy, np. prototyp	Projekt długotrwały – istnieje lub będzie utrzymywany przez długi czas
Projekt dla wąskiego grona użytkowników, np. do użytku wewnętrznego	Projekt dla szerokiego grona użytkowników, np. udostępniony publicznie w Internecie
Bezpieczeństwo i niezawodność nie są kluczowe, np. gra single player uruchamiana lokalnie	Bezpieczeństwo i niezawodność są kluczowe, np. serwer sieciowy, platforma dla innego oprogramowania

Tabela 1. Czy warto robić code review w danym projekcie?

JAK PODCHODZIĆ DO SPRAWY?

W tym artykule nie padną żadne konkretne przykłady kodu, konstrukcji w językach programowania, które należy promować lub których należy unikać. Toteż artykuł może być dla ciebie przydatny niezależnie od tego, w jakim języku programujesz. Zamiast tego zajmiemy się ogólnym podejściem do robienia review. Zacznijmy jednak od początku…

Podczas robienia code review, tak samo, jak podczas programowania, obowiązują **dobre praktyki programistyczne**. Z pewnością znasz wiele z nich, bardziej lub mniej sformalizowanych, obojętne, czy przechodziłeś/aś jakieś kursy w tym kierunku, studiowałeś/aś informatykę czy może jesteś samoukiem i praktykiem. Mam na myśli wszelkie zasady prawidłowego, wydajnego, jak i prostego używania dostępnych konstrukcji w danym języku i bibliotekach. Wszystko, co mielibyśmy stale z tyłu głowy, pisząc swój kod, warto też brać pod uwagę, przeglądając czyjś kod.

Obowiązują też oczywiście **zasady narzucone w danej organizacji**, w danym zespole czy projekcie. Jeżeli mamy spisany dokument typu „coding standard", który narzuca określony sposób nazywania zmiennych (CamelCase czy snake_case?) albo obsługi błędów w kodzie (rzucamy wyjątki, czy zwracamy numeryczny kod błędu?), to przestrzeganie takich zasad warto sprawdzać, robiąc review kodu. Taki dokument warto mieć, ponieważ może on rozwiązywać wiele niejasności i stanowić punkt odniesienia. Nawet jednak, kiedy żadnego formalnego standardu nie mamy spisanego, to w każdym zespole, który działa nie od dziś, z pewnością obowiązują pewne niepisane zasady (tzw. „wiedza plemienna"), które warto odkryć i się do nich dostosować.

Wreszcie, podczas robienia code review obowiązuje **kultura osobista**. Koniec końców komentarze w review to komunikacja człowieka z człowiekiem. Nie piszemy ich do kompilatora ani do ChatGPT, ale do żywej osoby. Wrócimy jeszcze później do tematu pisania komentarzy w review bardziej szczegółowo, ale warto zaznaczyć już teraz, że obowiązuje w nich (jak to się dawniej określało) „netykieta", czyli zasady uprzejmości i dobrego wychowania, które powinny obowiązywać w Internecie, a tak naprawdę zawsze i wszędzie. W przypadku code review szczególnie może się przydać znajomość sztuki dawania i otrzymywania „**feedbacku**", czyli informacji zwrotnej. Warto doczytać na ten temat.

W pewnym dużym projekcie obowiązywała zasada, że każda zmiana w kodzie musiała przejść review jednego programisty-leada, który trzymał pieczę nad całym kodem. Nazwijmy go „Wielki Ben". To on tworzył podwaliny tego projektu wiele lat temu, a teraz – kiedy jego utrzymanie zostało powierzone nowemu zespołowi – nadal pilnuje, aby wszystko było zgodne z jego pierwotną wizją. Każe on pisać rozwlekłe komentarze objaśniające nawet najbardziej oczywiste rzeczy wynikające z samego kodu. Zabrania używać inteligentnych wskaźników i innych nowoczesnych mechanizmów w języku programowania. Wszystko musi być nazwane tak, jak on chce. Nie ma z nim dyskusji na temat szczegółów ani miejsca na własną opinię. Nietrudno sobie wyobrazić, co programiści w tym zespole myślą o nim, jak i pracy w tym projekcie.

Powyższa historia jest całkowicie zmyślona, a wszelka zbieżność z realnymi osobami czy projektami jest przypadkowa. Jednakże nikt nie chciałby trafić do takiego projektu. Tym bardziej my, robiąc review czyjegoś kodu, nie chcemy być takim Wielkim Benem. Dlatego warto zacząć od podstaw i zastanowić się, jakie mentalne podejście byłoby dobre podczas robienia code review.

Jako pierwszą i najważniejszą zasadę można polecić, aby uczestnicząc w review po którejkolwiek ze stron, **pozbyć się własnego ego**. Brzmi to może jak duchowe mądrości jakiegoś guru z dalekiego wschodu albo hipisa, który wziął za dużo LSD, ale w praktyce oznacza rzecz bardzo prostą: aby zbytnio nie przywiązywać się do swojego kodu, a uwag na jego temat nie brać do siebie. Można powiedzieć: „Pamiętaj, twój kod to nie jesteś ty!"

Napisany przez nas kod nie istnieje sam dla siebie, ale ma czemuś służyć. Doświadczony programista wie, że często bardziej wartościowa dla projektu, w porównaniu z dodaniem całych setek linii nowego kodu, bywa modyfikacja jednej kluczowej linii. Jeszcze bardziej wartościowe, a przy tym satysfakcjonujące, bywa usuwanie tysięcy linii kodu, które stały się niepotrzebne. Krótszy kod to w końcu mniej zakamarków, w których mogą czaić się nieodkryte jeszcze błędy. Nie warto więc nadto przywiązywać się do linijek kodu przez nas dodanych.

Tu pojawia się pytanie o **odpowiedzialność**. Jeżeli nie powinniśmy przywiązywać się do napisanego przez nas kodu i zaciekle go bronić, to czy w ogóle nie powinniśmy go nazywać „swoim kodem", tylko traktować cały kod jako wspólny? Myślę, że przesadzić można

KLUCZ DO SUKCESU TESTÓW FUNKCJONALNYCH, CZYLI O DANYCH TESTOWYCH

INDEX: 285358

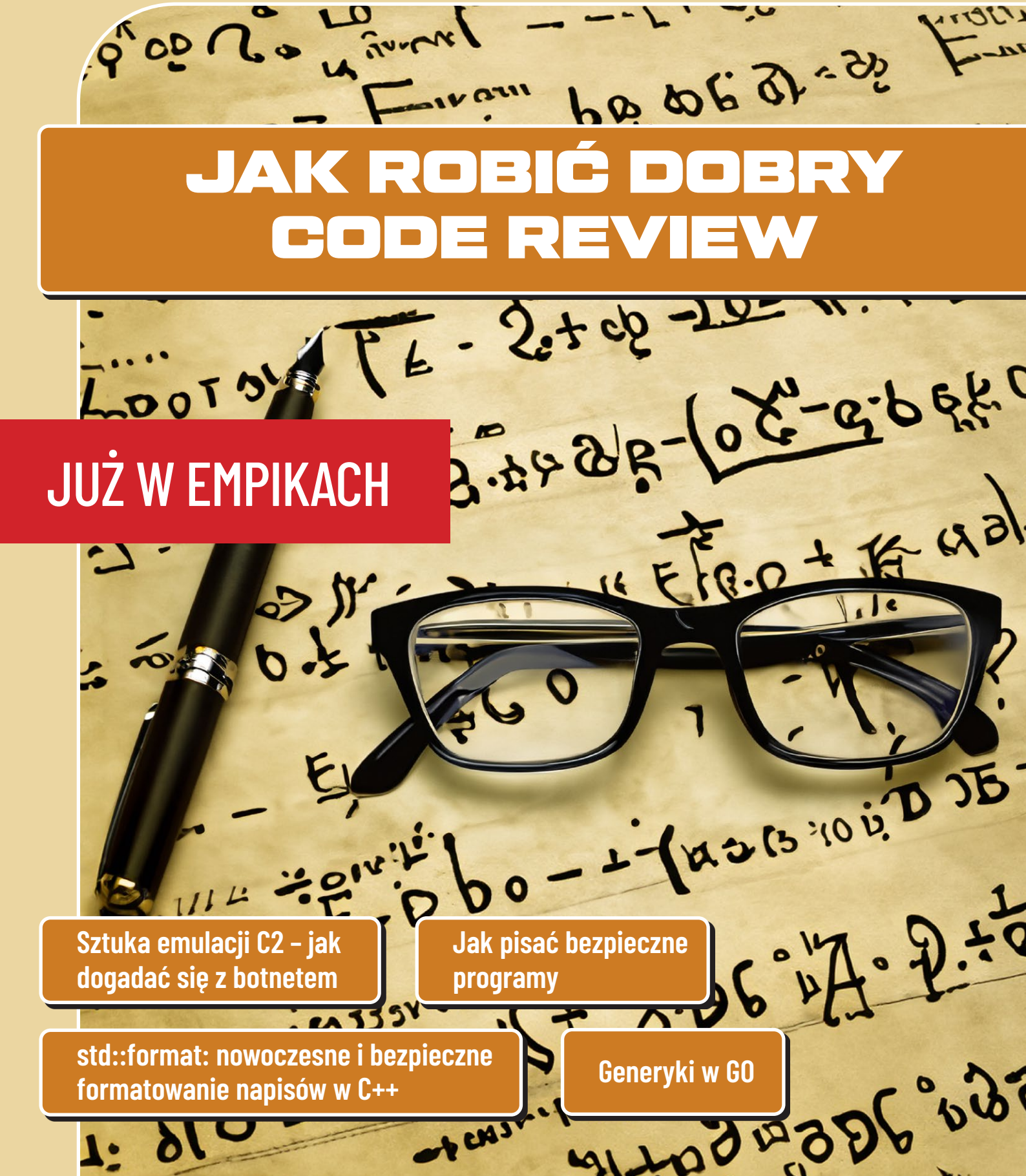
www.programistamag.pl

Magazyn programistów i liderów zespołów IT

programista

2/2024 (112)

Cena 28,90 zł (w tym VAT 8%)



JAK ROBIĆ DOBRY CODE REVIEW

JUŻ W EMPIKACH

Sztuka emulacji C2 – jak dogadać się z botnetem

Jak pisać bezpieczne programy

std::format: nowoczesne i bezpieczne formatowanie napisów w C++

Generyki w GO