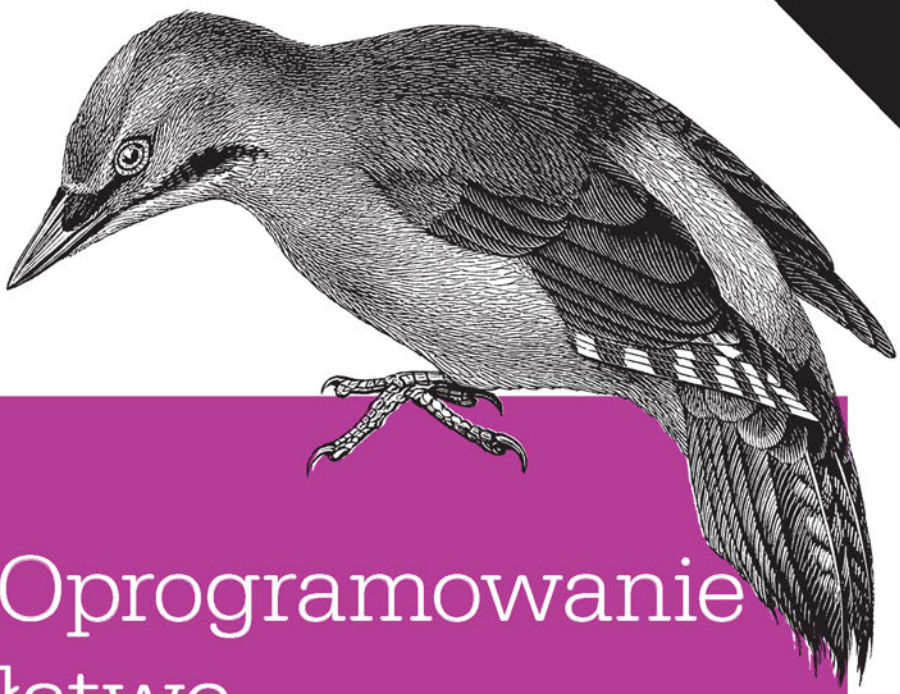
O'REILLY®

Edycja Java



Oprogramowanie łatwe w utrzymaniu

PISZ KOD PODATNY NA PRZYSZŁE ZMIANY

Helion

Joost Visser

Tytuł oryginału: Building Maintainable Software, Java Edition:
Ten Guidelines for Future-Proof Code

Tłumaczenie: Łukasz Suma

ISBN: 978-83-283-2843-3

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Spis treści

ROZDZIAŁ 1.

Wprowadzenie

„Kto napisał ten kod??? Nie mogę tak pracować!!!”

Dowolny programista

Praca programisty komputerowego to wspaniałe zajęcie. Gdy stajesz przed jakimś problemem i otrzymujesz odpowiednie wymagania, możesz opracować rozwiązanie i przełożyć je na język zrozumiały dla komputera. Jest to zadanie bardzo trudne, lecz dające wiele satysfakcji. Kariera programisty komputerowego bywa również jednak rzeczą wymagającą dużej pracowitości i poświęcenia. Jeśli często musisz wprowadzać zmiany w kodzie napisanym przez inne osoby (lub nawet przez samego siebie), wiesz doskonale, że może to być zajęcie naprawdę proste albo naprawdę trudne. Czasami jesteś w stanie szybko zidentyfikować linie kodu, które należy zmienić. Zmiana jest wyraźnie wyizolowana, a testy potwierdzają, że nowy kod działa dokładnie tak, jak chciałeś. Innym razem jedynym rozwiązaniem okazuje się zastosowanie hacka, który więcej problemów powoduje, niż rozwiązuje.

Prostota lub złożoność, jaka wiąże się z wprowadzaniem zmian w oprogramowaniu, jest określana jako jego **pielęgnowalność** lub **utrzymywalność** (ang. *maintainability*)[1]. Pielęgnowalność systemu oprogramowania jest determinowana przez właściwości jego kodu źródłowego. W niniejszej książce omówimy te właściwości oraz zaprezentujemy 10 wytycznych pomocnych w pisaniu kodu źródłowego, który będzie łatwo modyfikować.

[1] W literaturze można się również spotkać z terminami **naprawialność**, **serwisowalność**, **łatwość konserwacji**, **łatwość pielęgnacji** lub **łatwość utrzymania** — *przyp. tłum.*

W tym rozdziale wyjaśnimy, co rozumiemy pod pojęciem pielęgnowalności. Następnie omówimy powody, dla których jest ona tak ważną kwestią. To umożliwi nam wprowadzenie do głównego tematu tej książki, czyli tego, jak budować oprogramowanie, które jest łatwe w pielęgnacji od samego początku. Na koniec tego wprowadzenia omówimy powszechne zarzuty dotyczące pielęgnowalności oraz przedstawimy podstawowe zasady stojące za wspomnianymi 10 wytycznymi, które zostały zaprezentowane w niniejszej publikacji.

1.1. Czym jest pielęgnowalność?

Wyobraź sobie dwa różne systemy informatyczne, które oferują dokładnie tę samą funkcjonalność. Przy tych samych danych wejściowych wytwarzają ten sam efekt końcowy. Jeden z tych dwóch systemów działa szybko i jest prosty w obsłudze, a jego kod źródłowy łatwo jest zmodyfikować. Drugi jest powolny i trudno się go używa, a jego kodu źródłowego niemal nie da się zrozumieć, nie mówiąc już nawet o wprowadzeniu w nim jakichkolwiek zmian. Mimo że obydwa te systemy oferują te same możliwości funkcjonalne, ich jakość różni się w oczywisty sposób.

Pielęgnowalność (czyli to, jak łatwo można zmodyfikować system) jest jedną z cech jakości oprogramowania. Inną jest wydajność działania (czyli to, jak szybko system wytwarza efekt końcowy).

Międzynarodowy standard ISO/IEC 25010:2011 (który w tej książce nazywać będziemy po prostu ISO 25010[2]) wyróżnia osiem cech składających się na jakość oprogramowania: **pielęgnowalność**, **stabilność funkcjonalna**, **wydajność działania**, **kompatybilność**, **użyteczność**, **niezawodność**, **bezpieczeństwo** i **przenośność** (ang. odpowiednio: *maintainability*, *functional suitability*, *performance efficiency*, *compatibility*, *usability*, *reliability*, *security*, *portability*). W tej książce skupimy się wyłącznie na pielęgnowalności.

Choć standard ISO 25010 nie określa sposobu mierzenia jakości oprogramowania, nie znaczy to jeszcze, że pomiar taki jest niemożliwy. W dodatku A przedstawiliśmy, jak mierzymy jakość oprogramowania w ramach Software Improvement Group (SIG) w zgodzie z wytycznymi standardu ISO 25010.

[2] Jego pełna nazwa to: *International Standard ISO/IEC 25010. Systems and Software Engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and Software Quality Models*, wydanie pierwsze, 2011-03-01.

Cztery rodzaje działań związanych z pielęgnowaniem oprogramowania

Pielęgnacja oprogramowania nie polega na naprawianiu zużywających się elementów. Oprogramowanie nie ma charakteru fizycznego, a co za tym idzie, nie niszczy się samo z siebie, jak mają w zwyczaju przedmioty fizyczne. Mimo to większość systemów informatycznych jest nieustannie modyfikowana już po dostarczeniu ich klientom. Właśnie o to chodzi w pielęgnowaniu oprogramowania. Można tu wyróżnić cztery podstawowe rodzaje zadań:

- W oprogramowaniu wykrywane są błędy, które muszą być naprawiane. Tego typu działania nazywane są **pielęgnacją korekcyjną** (ang. *corrective maintenance*).
- System musi być dostosowywany do zmian występujących w środowisku, w którym działa. Przykładem mogą tu być aktualizacje systemu operacyjnego lub wykorzystywanej technologii. Nosi to nazwę **pielęgnacji adaptacyjnej** (ang. *adaptive maintenance*).
- Użytkownicy systemu (albo inni interesariusze) zmieniają wymagania lub zgłaszają nowe. Działania związane z ich wdrażaniem określane są mianem **pielęgnacji doskonalącej** (ang. *perfective maintenance*) lub po prostu **udoskonalania**.
- Znaleziono sposoby, aby poprawić jakość systemu lub zapobiec wystąpieniu przyszłych błędów w działaniu oprogramowania. Działania mające wprowadzić w życie tego rodzaju zmiany noszą nazwę **konserwacji profilaktycznej** (ang. *preventive maintenance*).

1.2. Dlaczego pielęgnowalność jest ważna?

Jak zdążyłeś się już dowiedzieć, pielęgnowalność jest zaledwie jedną z ośmiu charakterystyk jakości produktu programistycznego określonych w normie ISO 25010. Dlaczego zatem jest tak istotna, aby miała zasługiwać na napisanie całej książki poświęconej wyłącznie jej samej? Istnieją dwie odpowiedzi na to pytanie:

- Pielęgnowalność — albo też jej brak — ma znaczący wpływ na biznes.
- Pielęgnowalność ma decydujące znaczenie dla innych charakterystyk jakości.

Obydwie te odpowiedzi zostaną szerzej omówione w dwóch kolejnych punktach.

Pielęgnowalność ma znaczący wpływ na biznes

W procesie rozwoju oprogramowania etap pielęgnacji systemu trwa często 10 lat lub nawet dłużej. Przez większą część tego czasu mamy do czynienia z ciągłym strumieniem problemów, które należy rozwiązać (to pielęgnacja korekcyjna i adaptacyjna), a także propozycji udoskonaleń, które muszą być wdrożone (to pielęgnacja udoskonalająca). Wydajność i efektywność, z jakimi problemy mogą być rozwiązywane, a ulepszenia wprowadzane w życie, mają zatem duże znaczenie dla interesariuszy.

Wysiłki związane z pielęgnacją są mniejsze, gdy rozwiązania problemów i udoskonalenia mogą być wdrażane szybko i łatwo. Jeśli wydajna pielęgnacja pozwala ograniczyć liczbę odpowiedzialnych za nią osób (programistów), obniża również koszty operacyjne. Gdy liczba programistów nie ulega zmianie, dzięki wydajnej pielęgnacji mają oni więcej czasu na inne zadania, takie jak tworzenie nowej funkcjonalności. Szybkie wdrażanie ulepszeń oznacza skrócenie czasu wprowadzania na rynek nowych produktów oraz usług obsługiwanych przez system. Zarówno w przypadku rozwiązywania problemów, jak i wdrażania ulepszeń sprawdza się twierdzenie, że jeśli działania te są powolne i kłopotliwe, może to doprowadzić do sytuacji, w której nie uda się dotrzymać terminów lub system stanie się bezużyteczny.

Firma SIG zgromadziła empiryczne dowody, że rozwiązywanie problemów i wdrażanie udoskonaleń są działaniami dwukrotnie szybszymi w systemach o pielęgnowalności przewyższającej średnią niż w systemach z pielęgnowalnością niższą niż średnia. Dwukrotność to znacząca wartość w przypadku systemów biznesowych. Czas niezbędny do rozwiązania problemu i wdrożenia ulepszenia liczy się w dniach lub tygodniach. Nie mamy tu do czynienia z różnicą pomiędzy poprawieniem 5 a 10 błędów w ciągu godziny, lecz z różnicą pomiędzy byciem pierwszym, który dostarczy nowy produkt na rynek, a oglądaniem pleców konkurencji, która wyprzedza nas o całe miesiące.

A to tylko różnica pomiędzy pielęgnowalnością powyżej i poniżej średniej. W firmie SIG mieliśmy do czynienia z zupełnie nowymi systemami, których pielęgnowalność była tak niska, że właściwie nie dało się ich modyfikować — działo się tak nawet jeszcze przed wejściem tych systemów do produkcji. Zmiany wprowadzały więcej błędów, niż udawało się dzięki nim naprawić. Proces programowania trwał tak długo, że dawno już zdążyło się zmienić środowisko biznesowe (a więc i wymagania użytkownika). Konieczne były

kolejne modyfikacje, które wprowadzały jeszcze więcej błędów. Najczęściej okazywało się, że systemy takie musiały być spisane na straty, jeszcze zanim miały okazję doczekać się swojego pierwszego wydania.

Pielęgnowalność ma decydujące znaczenie dla innych charakterystyk jakości

Innym powodem, dla którego pielęgnowalność to szczególny aspekt jakości oprogramowania, jest to, że ma ona wpływ na inne charakterystyki jakości. Gdy system jest wysoce pielęgnowalny, łatwiej dokonywać ulepszeń w innych obszarach jakości, takich jak naprawianie błędów zabezpieczeń. Bardziej ogólnie rzecz ujmując, można stwierdzić, że optymalizacja systemu informatycznego wymaga modyfikacji jego kodu źródłowego, niezależnie od tego, czy chodzi o kwestie wydajności, przydatności funkcjonalnej, bezpieczeństwa czy też którąkolwiek inną z siedmiu pozostałych niepielęgnowalnościowych charakterystyk zdefiniowanych przez standard ISO 25010.

Zdarza się, że są to niewielkie, lokalne modyfikacje. Czasami wiążą się one z bardziej inwazyjną restrukturyzacją. Wszystkie zmiany wymagają odszukania określonego fragmentu kodu źródłowego, przeanalizowania go, zrozumienia jego wewnętrznej logiki oraz miejsca w procesie biznesowym, który oprogramowanie ma upraszczać, przeanalizowania zależności występujących pomiędzy różnymi częściami kodu, przetestowania ich oraz przepchnięcia przez cały proces programistyczny. W każdym przypadku modyfikacje te łatwiej wykonać w systemie pielęgnowalnym, dzięki czemu implementacja optymalizacji jakościowych może odbyć się szybciej i efektywniej. Na przykład, wysoce pielęgnowalny kod jest stabilniejszy niż kod niepielęgnowalny, ponieważ zmiany w systemie pielęgnowalnym mają mniej efektów ubocznych niż zmiany w systemie powikłanym, który trudno się analizuje i testuje.

1.3. Trzy podstawowe zasady, na których oparto wytyczne umieszczone w tej książce

Skoro pielęgnowalność jest tak istotna, jak możesz poprawić ją w przypadku kodu, który piszesz? W książce tej przedstawiono 10 wytycznych, które — gdy będzie się ich przestrzegać — umożliwiają tworzenie wysoce pielęgnowalnego kodu. W kolejnych rozdziałach każda z tych wytycznych jest prezentowana

i omawiana szczegółowo. W niniejszym rozdziale przedstawimy zasady stojące za tymi wytycznymi. Oto one:

1. Pielęgnowalność wzrasta, gdy trzymamy się prostych wytycznych.
2. Pielęgnowalność nie wynika z refleksji, lecz powinna być brana pod uwagę od samego początku projektu programistycznego. Liczy się tu dosłownie każdy składnik.
3. Niektóre naruszenia są gorsze od innych. Im bardziej przy tworzeniu systemu informatycznego trzymamy się wytycznych, tym bardziej jest on pielęgnowalny.

Zasady te zostały szczegółowo wyjaśnione poniżej.

Zasada 1. Pielęgnowalność zyskuje najbardziej dzięki prostym wytycznym

Niektórym może się wydawać, że pielęgnowalność wymaga użycia jakiejś „srebrnej kuli", czyli jednej technologii lub zasady, która załatwia temat raz na zawsze, **automagicznie**. Kierujemy się tu zupełnie inną zasadą, wierząc, że pielęgnowalność wymaga postępowania według prostych, naprawdę niewyszukanych wytycznych. Zapewniają one wystarczający, nie zaś doskonały poziom pielęgnowalności (czymkolwiek ten ostatni by nie był). Kod źródłowy powstały według tych wytycznych można nadal poprawić, aby stał się on jeszcze bardziej pielęgnowalny. Jednak w pewnym momencie dodatkowe zyski płynące ze wzrostu pielęgnowalności stają się coraz mniejsze, podczas gdy koszt jej zwiększania szybko rośnie.

Zasada 2. Pielęgnowalność nie wynika z refleksji i liczy się każdy składnik

Pielęgnowalność powinna być uwzględniana w projekcie informatycznym od samego początku. Rozumiemy, że trudno stwierdzić, czy pojedyncze „naruszenie" wytycznych przedstawionych w tej książce ma wpływ ma całkowitą pielęgnowalność systemu. To właśnie dlatego *wszyscy* programiści muszą zachować dyscyplinę i podążać za wytycznymi, aby móc opracować system, który jest ogólnie pielęgnowalny. Z tego powodu Twój indywidualny wkład ma zasadnicze znaczenie dla całości.

Przestrzeganie wytycznych przedstawionych w tej książce nie tylko skutkuje otrzymaniem bardziej pielęgnowalnego kodu, lecz również daje właściwy przykład Twoim kolegom programistom. Pozwala to uniknąć „efektu wybitych okien", powstającego, gdy inni programiści chwilowo poluzowują swoją dyscyplinę i idą na skróty. W świeceniu dobrym przykładem nie chodzi koniecznie o to, aby być najbardziej wykwalifikowanym inżynierem, lecz raczej o to, aby zachować właściwą dyscyplinę w czasie programowania.

Pamiętaj, że nie piszesz kodu wyłącznie dla siebie, lecz również dla mniej doświadczonych programistów, którzy przyjdą po Tobie. Ta myśl powinna pomóc Ci upraszczać rozwiązanie, które tworzysz.

Zasada 3. Niektóre naruszenia są gorsze od innych

Wytyczne przedstawione w tej książce traktują progi metryczne jako regułę bezwzględną. W rozdziale 2. mówimy na przykład, abyś nigdy nie pisał metod, które zawierają więcej niż 15 linii kodu. Jesteśmy w pełni świadomi, że w praktyce niemal zawsze będą się zdarzały wyjątki w stosowaniu tej wytycznej. Co zaś, gdy w jakimś fragmencie kodu uda Ci się naruszyć jakąś wytyczną lub nawet kilka z nich? Wiele narzędzi programistycznych pomagających w zachowaniu właściwej jakości oprogramowania działa zgodnie z założeniem, że każde, dosłownie każde naruszenie zasad jest złe. Ukrytym założeniem jest tu to, że wszystkie tego rodzaju naruszenia powinny być usunięte. W praktyce usuwanie wszystkich naruszeń nie jest ani konieczne, ani opłacalne. Takie bezkompromisowe podejście do naruszeń wytycznych może doprowadzić do sytuacji, w której programiści będą zupełnie ignorować naruszenia.

Nasze podejście jest inne. Aby metryki były proste, a jednocześnie praktyczne, określamy jakość pełnej bazy kodu nie na podstawie liczby naruszeń, z którymi mamy w niej do czynienia, lecz według jej **profili jakości** (ang. *quality profiles*). Profil jakości dzieli metryki na odrębne kategorie, obejmujące przypadki od kodu całkowicie zgodnego z wytycznymi aż do takiego, który silnie je narusza. Korzystając z profili jakości, możemy odróżnić umiarkowane naruszenia (takie jak metoda zawierająca 20 linii kodu) od poważnych (przykładem może tu być metoda z 200 liniami kodu). W kolejnych podrozdziałach (tuż po podrozdziale następnym, w którym omówione zostały powszechne nieporozumienia związane z pielęgnowalnością) wyjaśnimy, w jaki sposób profile jakości są wykorzystywane do mierzenia pielęgnowalności systemu.

1.4. Nieporozumienia związane z pielęgnowalnością

W podrozdziale tym omówimy pewne nieporozumienia związane z pielęgnowalnością, z którymi można się często spotkać w praktyce.

Nieporozumienie: Pielęgnowalność jest zależna od języka programowania

„W naszym systemie zastosowano najnowocześniejszy język programowania. Z tego powodu jest on co najmniej tak pielęgnowalny, jak każdy inny system".

Dane, które udało nam się zebrać w firmie SIG, nie wskazują, aby technologia (język programowania) wybrana dla systemu była dominującym czynnikiem determinującym pielęgnowalności. Nasz zbiór danych obejmuje systemy opracowane w języku Java, które należą do najbardziej pielęgnowalnych, lecz również takie, które są pośród najtrudniejszych w pielęgnacji. Średnia pielęgnowalność wszystkich systemów javowych w naszym porównaniu jest po prostu średnia, tak samo jest zresztą w przypadku C#. Pokazuje to, że możliwe jest opracowanie bardzo pielęgnowalnych systemów w Javie (oraz w C#), ale zastosowanie któregokolwiek z tych języków programowania nie gwarantuje w żaden sposób wysokiego poziomu pielęgnowalności systemu. Najwyraźniej na pielęgnowalność mają wpływ inne czynniki.

Aby zachować spójność treści, w całej książce korzystamy z przykładowych fragmentów kodu napisanych w Javie. Jednak opisane tu wytyczne nie są specyficzne dla tego języka. W rzeczywistości firma SIG sprawdziła systemy napisane w ponad stu różnych językach programowania na podstawie wytycznych i metryk zawartych w tej książce.

Nieporozumienie: Pielęgnowalność jest zależna od branży

„Mój zespół zajmuje się pisaniem oprogramowania wbudowanego dla przemysłu motoryzacyjnego. Pielęgnowalność to zupełnie inny temat w tej branży".

Wierzymy, że wytyczne przedstawione w tej książce mogą być zastosowane do każdego rodzaju twórczości programistycznej: oprogramowania wbudowanego, gier komputerowych, oprogramowania naukowego, komponentów programistycznych, takich jak kompilatory i silniki baz danych, a także oprogramowania administracyjnego. Istnieją oczywiście różnice pomiędzy tymi dziedzinami.

Na przykład, w przypadku oprogramowania naukowego używa się często języka programowania specjalnego przeznaczenia, takiego jak język R wykorzystywany w analizach statystycznych. Ale również w R dobrze jest tworzyć krótkie i proste jednostki kodu. Oprogramowanie wbudowane musi działać w środowisku, w którym przewidywalność wydajności ma kluczowe znaczenie, a zasoby są ograniczone. Zatem w każdym przypadku, gdy trzeba wypracować kompromis pomiędzy wydajnością a pielęgnowalnością, ta pierwsza musi wygrać z drugą. Niezależnie jednak od dziedziny, z którą mamy do czynienia, charakterystyki określone przez standard ISO 25010 nadal mają zastosowanie.

Nieporozumienie: Pielęgnowalność jest tym samym, co brak błędów

„Powiedziałeś, że system musi mieć pielęgnowalność powyżej średniej. Jednak okazało się, że jest pełen błędów!".

Według definicji standardu ISO 25010 system może być wysoce pielęgnowalny, ale nadal cierpieć na braki w zakresie innych charakterystyk jakości. A zatem może mieć ponadprzeciętną pielęgnowalność i w dalszym ciągu sprawiać kłopoty związane ze stabilnością funkcjonalną, wydajnością, niezawodnością itd. Ponadprzeciętna pielęgnowalność nie oznacza nic więcej niż tylko to, że modyfikacje niezbędne, aby zredukować liczbę błędów, mogą być wykonane bardzo wydajnie i skutecznie.

Nieporozumienie: Pielęgnowalność ma charakter zerojedynkowy

„Mój zespół wielokrotnie był w stanie poprawiać błędy w tym systemie. Zatem udało się udowodnić, że jest on łatwy w pielęgnacji".

To rozróżnienie jest bardzo istotne. Pielęgnowalność to dosłownie możliwość pielęgnacji, utrzymywania czy też konserwacji. Zgodnie z definicją określoną przez standard ISO 25010 pielęgnowalność kodu źródłowego nie jest wielkością binarną. Jest za to miarą stopnia, w jakim zmiany mogą być zrobione w sposób wydajny i skuteczny. Pytanie, które należy tu zatem zadać, nie powinno dotyczyć tego, czy zmiany (takie jak poprawki błędów) zostały zrobione, lecz raczej tego, jaki wiązał się z tym wysiłek (wydajność) i czy dało to właściwy efekt (skuteczność).

Biorąc pod uwagę definicję pielęgnowalności według standardu ISO 25010, można by stwierdzić, że system informatyczny nigdy nie może być doskonale pielęgnowalny ani doskonale *nie*pielęgnowalny. W praktyce w firmie SIG zdarzało nam się trafiać na systemy, które można uznać za niepielęgnowalne. Systemy te charakteryzowały się tak niskim stopniem efektywności i skuteczności modyfikacji, że ich właścicieli po prostu nie było stać na ich pielęgnację.

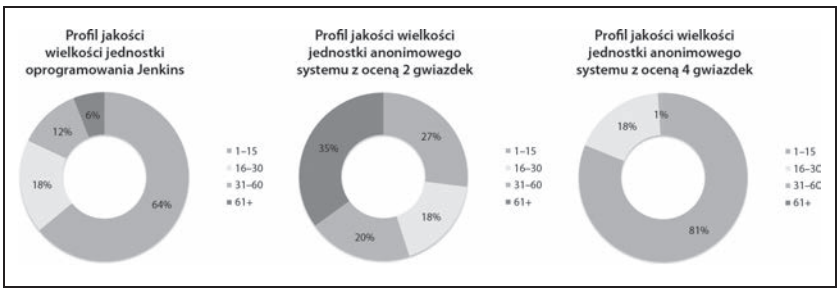
1.5. Ocena pielęgnowalności

Wiemy już, że pielęgnowalność to charakterystyka jakości odniesiona do pewnej skali. Oznacza ona różne stopnie możliwości pielęgnacji systemu. Co jednak oznacza „łatwy w pielęgnacji", a co „trudny w pielęgnacji"? Skomplikowany system będzie, rzecz jasna, łatwiej pielęgnować ekspertowi niż mniej doświadczonemu programiście. Tworząc nasze porównania w firmie SIG, pozwalamy odpowiedzieć na to pytanie metrykom stosowanym w branży oprogramowania. Jeśli metryka systemu plasuje się poniżej średniej, to jest on trudniejszy w pielęgnacji niż przeciętny. Punkt odniesienia jest rekalibrowany rokrocznie. Jako że branża uczy się kodować bardziej efektywnie (na przykład dzięki pomocy nowych technologii), średnia dla metryk dąży do poprawy wraz z czasem. To, co było normą w inżynierii oprogramowania kilka lat temu, w tej chwili może odstawać jakościowo. Punkt odniesienia odzwierciedla zatem aktualny stan wiedzy w inżynierii oprogramowania.

SIG w ramach swojego rankingu stosuje wobec systemów ocenę wyrażoną za pomocą liczby gwiazdek, przy czym jedna gwiazdka oznacza tu system najtrudniejszy w pielęgnacji, a pięć — system najłatwiejszy. Rozkład ocen systemów w tej skali od 1 do 5 gwiazdek przedstawia się następująco: 5%-30%-30%-30%-5%. Oznacza to, że w rankingu systemy należące do najlepszych 5% otrzymują ocenę 5 gwiazdek. W systemach tych wciąż występują pewne naruszenia wytycznych, jest ich jednak znacznie mniej niż w systemach ocenionych gorzej.

Ocena określona w gwiazdkach stanowi prognozę rzeczywistej pielęgnowalności systemu. Firmie SIG udało się zgromadzić dowody empiryczne na to, że rozwiązywanie problemów i wdrażanie ulepszeń jest dwukrotnie szybsze w przypadku systemów, które otrzymały ocenę 4 gwiazdek, niż w systemach ocenionych na 2.

Systemy w rankingu są uszeregowane na podstawie ich metrycznych profili jakości. Na rysunku 1.1 przedstawione zostały trzy przykłady profili jakości wielkości jednostki (czytelnicy papierowego wydania książki mogą obejrzeć kolorowe wersje tego i kolejnych rysunków, korzystając z repozytorium opracowanego dla potrzeb publikacji: *ftp://ftp.helion.pl/przyklady/oplaut.zip*).



Rysunek 1.1. Przykład trzech profili jakości

Pierwszy wykres na rysunku 1.1 przedstawia profil jakości wielkości jednostki kodu źródłowego oprogramowania Jenkins w wersji 1.625, będącego popularnym serwerem ciągłej integracji dostępnym na zasadach otwartego kodu (rozwiązanie dostępne jest pod adresem: *https://jenkins.io/index.html*). Widoczny tu profil jakości informuje nas, że w bazie kodu Jenkinsa 64% kodu znajduje się w metodach, których długość nie przekracza 15 linii (czyli jest zgodna z odpowiednimi wytycznymi). Profil ten pokazuje również, że 18% całego kodu znajduje się w metodach, których długość zawiera się w przedziale od 15 do 30 linii, a 12% — w metodach o długości od 31 do 60 linii kodu. Baza kodu Jenkinsa nie jest doskonała. Występują w niej poważne naruszenia wytycznych dotyczących wielkości jednostki: 6% kodu znajduje się w bardzo dużych jednostkach (takich, których długość przekracza 60 linii kodu).

Drugi z wykresów zaprezentowanych na rysunku 1.1 przedstawia profil jakości systemu, który otrzymał ocenę 2 gwiazdek. Zauważ, że ponad 1/3 bazy kodu znajduje się w jednostkach, których długość przekracza 60 linii. Pielęgnacja tego systemu będzie wymagała dużego nakładu pracy.

Trzeci i ostatni wykres widoczny na rysunku 1.1 przedstawia charakterystykę wielkości jednostki dla systemu ocenionego na 4 gwiazdki. Porównaj ten wykres z pierwszym. Możesz stwierdzić, że Jenkins spełnia wytyczne dotyczące wielkości jednostki, które umożliwiają przyznanie mu oceny 4 gwiazdek

(choć już nie 5), ponieważ procentowe udziały jego kodu w każdej z kategorii są niższe niż limity wyznaczone dla tej oceny.

W ramkach znajdujących się na końcu wszystkich rozdziałów poświęconych poszczególnym wytycznym przedstawiamy kategorie profili jakości związanych z tymi wytycznymi w takiej formie, w jakiej wykorzystywane są w SIG do oceny pielęgnowalności. A dokładnie: prezentujemy tam limity oraz maksymalny odsetek kodu, który może należeć do każdej z kategorii, aby mógł on otrzymać ocenę 4 gwiazdek lub wyższą (a więc należeć do najlepszych 35% rankingu).

1.6. Przegląd wytycznych dotyczących pielęgnowalności

W kolejnych rozdziałach szczegółowo przedstawione zostaną poszczególne wytyczne, tutaj jednak postanowiliśmy wymienić wszystkie 10, aby dać Ci możliwość szybkiego zapoznania się z całą ich listą. Zalecamy rozpocząć lekturę od rozdziału 2., a następnie czytać następne rozdziały w kolejności, w której występują w książce.

„Pisanie krótkich jednostek kodu" (rozdział 2.)
: Krótsze jednostki (czyli metody i konstruktory) łatwiej się analizuje, testuje i ponownie wykorzystuje.

„Pisanie prostych jednostek kodu" (rozdział 3.)
: Jednostki zawierające mniejszą liczbę punktów decyzyjnych łatwiej analizować i testować.

„Pisanie kodu jeden raz" (rozdział 4.)
: Zawsze należy unikać powielania kodu źródłowego, ponieważ zmiany będą musiały być wprowadzane do każdej z kopii. Powielanie jest również źródłem błędów regresji.

„Ograniczanie wielkości interfejsów jednostek" (rozdział 5.)
: Jednostki (metody i konstruktory) z mniejszą liczbą parametrów łatwiej się testuje i wykorzystuje ponownie.

„Separowanie zagadnień w modułach" (rozdział 6.)
: Luźno sprzężone moduły (klasy) łatwiej jest modyfikować, umożliwiają też one tworzenie bardziej modułowych systemów.

„Luźne sprzęganie komponentów architektonicznych" (rozdział 7.)
: Wysokopoziomowe komponenty systemu, które są ze sobą luźno sprzężone, łatwiej jest modyfikować, a ponadto umożliwiają one tworzenie bardziej modułowych systemów.

„Równoważenie architektury komponentów" (rozdział 8.)
: Zrównoważona architektura, w której nie występuje zbyt wiele ani zbyt mało komponentów i w której mają one jednolitą wielkość, ma najbardziej modułowy charakter i umożliwia prostą modyfikację poprzez separację zagadnień.

„Ograniczanie wielkości bazy kodu" (rozdział 9.)
: Wielki system jest trudny w pielęgnacji, ponieważ analizować, zmieniać i testować należy w jego przypadku więcej kodu. Ponadto wydajność pielęgnacji *w przeliczeniu na linię kodu* jest niższa w dużych systemach niż w małych.

„Automatyzowanie testów" (rozdział 10.)
: Zautomatyzowane testy (to jest takie, które da się przeprowadzić bez interwencji człowieka) zapewniają niemal natychmiastowe uzyskiwanie informacji zwrotnych na temat skuteczności wprowadzanych zmian. Ręczne testy są mało skalowalne.

„Pisanie czystego kodu" (rozdział 11.)
: Pozostawianie w bazie kodu nieistotnych artefaktów, takich jak opisy, co jeszcze należy zrobić, a także martwego kodu sprawia, że nowym członkom zespołu trudniej jest wydajnie pracować. To z kolei oznacza zmniejszenie wydajności pielęgnowania kodu.

Skorowidz

L

Ł

M

N

O

P

R

S

Ś

T

U

W

Z

Ź

O'REILLY®

Profesjonalny programista zawsze pisze kod najwyższej jakości!

Oprogramowanie po wdrożeniu w środowisku produkcyjnym dalej wymaga opieki programisty. Aktualizacje, dostosowanie do zmian, udoskonalenia i usuwanie usterek — te czynności są warunkiem utrzymania systemu w dobrej kondycji. Niestety, jeśli twórca oprogramowania nie przestrzegał pewnych zasad, pielęgnacja kodu jest uciążliwa, nieefektywna, a bywa nawet, że niemożliwa. System przestaje działać ze wszystkimi tego konsekwencjami.

Aby tego uniknąć, wystarczy na etapie tworzenia kodu uwzględniać potrzebę jego utrzymywania w przyszłości. Niniejsza książka jest lekturą obowiązkową dla wszystkich, którzy chcą tworzyć łatwy w pielęgnacji kod. Przedstawiono tu 10 wytycznych prowadzących do tego celu. Zostały one gruntownie omówione, a ich znaczenie i sposób stosowania w praktyce wyjaśniono, posługując się przykładowymi fragmentami kodu. Napisano go w Javie, jednak książka okaże się przydatna również dla programistów używających innych języków.

W książce przedstawiono następujące zagadnienia:

- pielęgnacja kodu i jej znaczenie dla poprawnego działania systemu
- pielęgnowalność kodu i sposoby jej oceny
- 10 wytycznych tworzenia łatwego w pielęgnacji kodu
- wskazówki i wyjaśnienia dotyczące stosowania wytycznych w praktyce

Joost Visser — jest profesorem na Uniwersytecie im. Radbouda w Nijmegen. Zajmuje się programowaniem generycznym, zieloną informatyką, a także jakością i ewolucją oprogramowania.

Pascal van Eck — zajmuje się jakością oprogramowania. Jest autorem ponad 80 publikacji dotyczących bezpieczeństwa IT i metryk oprogramowania.

Rob van der Leek — jest konsultantem do spraw jakości oprogramowania. Bierze również udział w tworzeniu narzędzi do analizy programów.

Sylvan Rigal — zajmuje się jakością oprogramowania i prowadzi szkolenia z analizy ryzyka bezpieczeństwa programów.

Gijs Wijnholds — pracuje nad jakością oprogramowania w administracji publicznej. Jest ekspertem od Haskella i lingwistyki matematycznej.

ISBN 978-83-283-2843-3

9 788328 328433

Informatyka w najlepszym wydaniu

cena: 39,90 zł