O'REILLY®

Helion

Uporządkowany kod

Ćwiczenia z empirycznego projektowania oprogramowania

Kent Beck

Tytuł oryginału: Tidy First?: A Personal Exercise in Empirical Software Design

Tłumaczenie: Grzegorz Werner

ISBN: 978-83-289-1334-9

Authorized Polish translation of the English edition of *Tidy First?*
ISBN 9781098151249 © 2024 Kent Beck.

Helion S.A.
ul. Kościuszki 1c, 44-100 Gliwice
tel. 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *https://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
https://helion.pl/user/opinie/upkocw
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 13.

Jedna sterta

Czasem czytasz kod, który został podzielony na wiele małych części, ale w niezrozumiały sposób. Wrzuć te fragmenty na jedną wielką stertę i dopiero wtedy zacznij porządkowanie.

Największym kosztem kodu jest jego czytanie i rozumienie, a nie pisanie. Filozofia „zaczynania od porządków" preferuje wiele małych elementów, zarówno z perspektywy teoretycznej, w celu ograniczenia sprzężenia przez zwiększenie kohezji, jak i praktycznej, w celu zmniejszenia ilości szczegółów, o których musisz jednocześnie myśleć.

Preferencyjne traktowanie małych części ma pomagać w zrozumieniu kodu kawałek po kawałku. Czasem jednak proces ten zawodzi. Sposób interakcji między małymi częściami sprawia, że kod jest *mniej* zrozumiały. Aby odzyskać klarowność, trzeba najpierw zlepić kod, a następnie wyodrębnić z niego nowe, łatwiejsze do zrozumienia części.

Oto niektóre symptomy problemu:

- długie, powtarzające się listy argumentów;
- powtarzający się kod, zwłaszcza powtarzające się instrukcje warunkowe;
- złe nazewnictwo procedur pomocniczych;
- współdzielone, zmienne struktury danych.

Ponieważ zwykle preferujemy wiele mniejszych części, tworzenie jednej sterty podczas porządkowania wydaje się dziwne. Jest jednak osobliwie satysfakcjonujące. Próbuję zrozumieć kod w kawałkach. Zaczynam wątpić w swoje umiejętności. Robię zwrot o 180 stopni i zaczynam zlepiać wszystko w całość (bardzo pomagają w tym automatyczne refaktoryzacje, ale jeśli trzeba, robię to ręcznie). Co za ulga!

W miarę jak sterta rośnie, w moim umyśle zaczyna wyłaniać się pewien kształt. Widzę: najpierw liczymy to, a potem używamy tego, żeby obliczyć tamto! Dlaczego od razu tego nie powiedzieli? Teraz mogę sobie zadać tytułowe pytanie: czy powinienem najpierw posprzątać? Czy po prostu wprowadzić zmianę, która obecnie wydaje się jasna?

ROZDZIAŁ 14.

Wyjaśniające komentarze

Znasz ten moment, kiedy czytasz jakiś kod i mówisz sobie: „Aha, a więc *o to* chodzi!”? To cenny moment. Udokumentuj go.

Zapisz tylko to, co nie wynika w oczywisty sposób z kodu. Postaw się na miejscu przyszłego czytelnika albo samego siebie sprzed 15 minut. Co chciałbyś wtedy wiedzieć? Możesz napisać coś w rodzaju: „Poniższy kod komplikuje to, że trzeba jak najbardziej ograniczyć liczbę wywołań sieciowych”.

Pisz do kogoś konkretnego, nawet jeśli ten ktoś nie bardzo przypomina Ciebie. Czy jesteś jedynym biologiem w zespole informatyków? W takim razie lepiej wyjaśnij każdy biologiczny kontekst w kodzie, nawet jeśli Tobie wydaje się oczywisty. Chodzi o to, aby myśleć z perspektywy kogoś innego i spróbować uprzedzić ewentualne pytania.

Jeśli napotkasz plik bez nagłówka z komentarzem, rozważ dodanie nagłówka, który wyjaśni przyszłym czytelnikom, dlaczego przejrzenie tego pliku może być użyteczne. (Dziękuję za tę radę Allanowi Mertnerowi).

Dobrym momentem na dodanie komentarza jest zauważenie jakiegoś defektu. Na przykład: *// W razie dodania kolejnego przypadku trzeba zmienić ../foo*. Lepiej nie mieć takiego sprzężenia w kodzie. Prędzej czy później będziesz musiał nauczyć się, jak je wyeliminować, ale tymczasem lepiej dodać komentarz, który wskazuje problem ze sprzężeniem, zamiast zostawiać je zagrzebane w piasku.

ROZDZIAŁ 15.

Usuwanie zbędnych komentarzy

Kiedy widzisz komentarz, który mówi dokładnie to samo co kod, usuń go.

Celem kodu jest wyjaśnienie innym programistom, czego oczekujesz od komputera. Komentarze i kod prezentują różne kompromisy dla Ciebie jako autora i dla przyszłych czytelników. W komentarzach możesz wyjaśnić wszystko, co tylko zechcesz. Z drugiej strony, nie ma żadnego mechanizmu, który sprawdzałby dokładność komentarzy po zmianach w systemie, a niektóre komentarze stają się zbędne w miarę ewolucji kodu.

Niektórzy traktują śmiertelnie poważnie przykazanie komunikacji, dogmatycznie trzymając się reguł takich jak ta, która mówi, że każda procedura powinna być skomentowana. Prowadzi to do komentarzy podobnych do poniższego:

```
getX()
    # Zwracamy X
    return X
```

Ten komentarz dodaje koszty bez żadnych korzyści. Jako autor właśnie zmarnowałeś czas czytelnika — czas, którego już nie odzyska. Jeśli komentarz jest całkowicie zbędny, usuń go.

Porządki często się łączą. Poprzednie porządkowanie mogło sprawić, że komentarz stał się zbędny. Na przykład pierwotny kod mógł wyglądać tak:

```
if (generator)
    ...kod konfigurujący generator...
else
    # Brak generatora, zwracamy generator domyślny
    return getDefaultGenerator()
```

Po porządkach polegających na dodaniu klauzuli strażniczej kod wygląda tak:

```
if (! generator)
    # Brak generatora, zwracamy generator domyślny
```

```
        return getDefaultGenerator()
    ...kod konfigurujący generator...
```

Komentarz początkowo nie był zbędny. Zwracał naszą uwagę na bieżący kontekst (brak generatora) po przeczytaniu iluś wierszy kodu w innym kontekście (generator obecny, wymaga skonfigurowania). Jednak po porządkowaniu komentarz ten po prostu powtarza to, co mówi kod. Usuńmy go zatem. *Hasta la vista, auf wiedersehen, pa, pa*.

O łączeniu porządków powiem więcej w części II.


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

To ważna książka na ważny temat!

Dave Farley, założyciel i dyrektor Continuous Delivery Ltd.

Niechlujny kod to koszmar programisty. Utrudnia zrozumienie logiki programu i często prowadzi do problemów z debugowaniem. Komplikuje modyfikację i rozbudowę programu, pogarsza współpracę z zespołem. Z kolei uporządkowany kod jest zrozumiały i łatwy w utrzymaniu. To proste: czysty kod to szczęśliwy programista!

Ta zwięzła publikacja przyda się profesjonalistom, którzy lubią drobne ulepszenia prowadzące do dużych korzyści. Zrozumiale wyjaśniono w niej, na czym polega proces tworzenia czystego i niezawodnego kodu. W rozsądnej dawce podano zagadnienia teoretyczne, takie jak sprzężenie, kohezja, zdyskontowane przepływy pieniężne i opcjonalność. Porządkowanie kodu jest tu przedstawione jako element codziennej pracy programisty, prowadzący do poprawy struktury całego projektu. W książce znalazło się mnóstwo praktycznych przykładów, dzięki którym można wypróbować wybrane techniki, najlepiej sprawdzające się w danym przypadku.

Tę książkę polecam każdemu, komu zależy na czystym i czytelnym kodzie!

Gergely Orosz, autor newslettera *The Pragmatic Engineer*

Najciekawsze zagadnienia:

- teoretyczne podstawy projektowania oprogramowania
- różnica między zmianami działania systemu a zmianami jego struktury
- najlepszy czas na sprzątanie kodu
- dokonywanie dużych zmian małymi krokami
- projektowanie oprogramowania jako ćwiczenie z obszaru relacji międzyludzkich

Kent Beck jest programistą, twórcą programowania ekstremalnego i pionierem wzorców oprogramowania. Jest też sygnatariuszem *Manifestu Agile*. Mieszka w San Francisco w Kalifornii, pracuje na stanowisku głównego badacza w firmie Mechanical Orchard.

Helion
helion.pl
HELION S.A.
ul. Kościuszki 1c
44-100 Gliwice
tel.: 32 230 98 63
helion@helion.pl
KOD KORZYŚCI
Sięgnij po więcej!
ISBN 978-83-289-1334-9
9 788328 913349
Cena: 49,90 zł