Bogusław Krawczuk

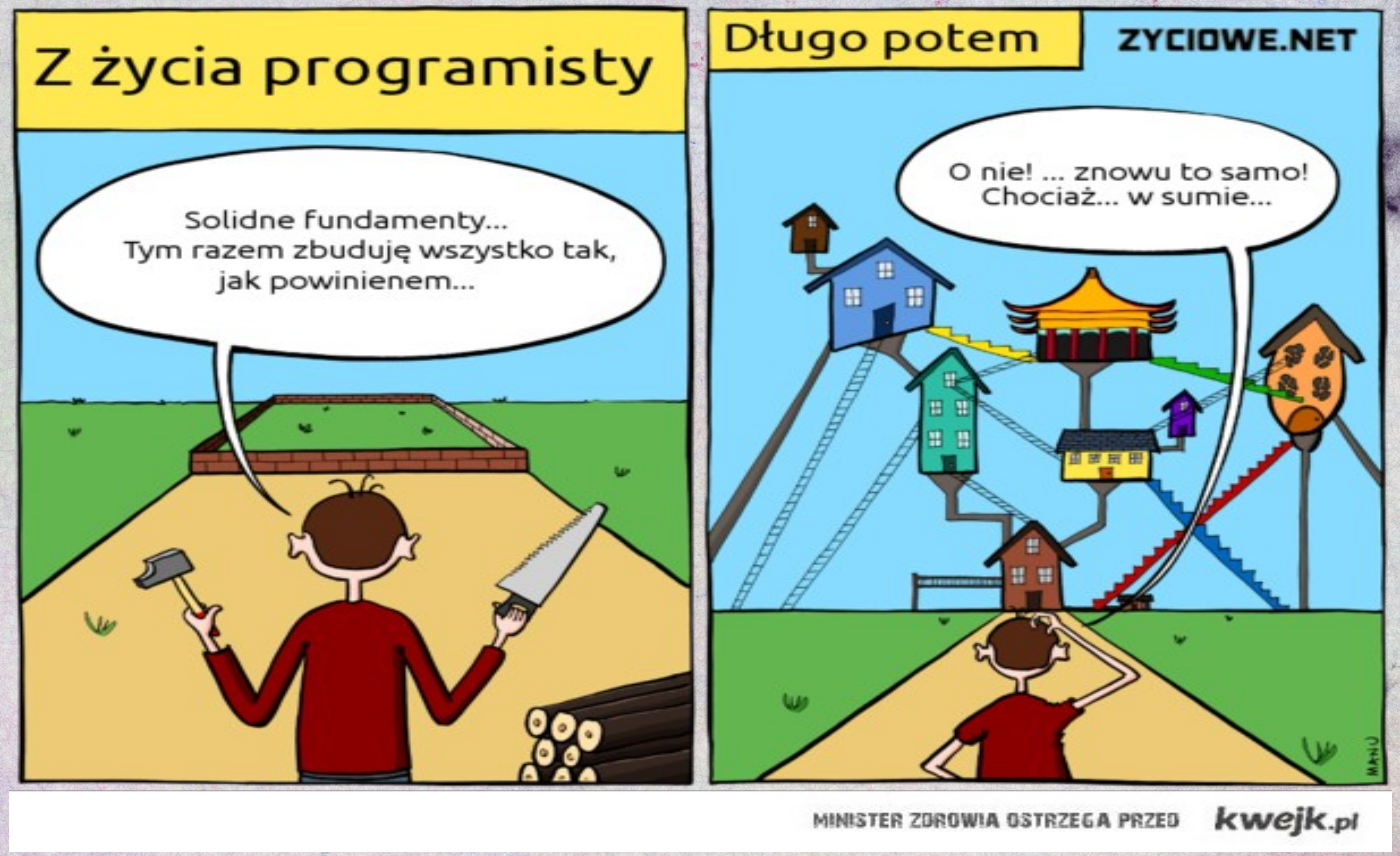
REFACTORING

czyli poprawianie kodu bez zmiany funkcjonalności

Motto:

prawo Hofstadtera:
Wszystko zajmuje więcej czasu, niż się spodziewałeś, nawet jeżeli weźmiesz pod uwagę prawo Hofstadtera.

Dlaczego potrzebujemy testowania i refaktoringu?



Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Programowanie jest rozkoszą, debugowanie jest koszmarem."*

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Jeśli debugowanie jest procesem usuwania błędów z oprogramowania, to programowanie musi być procesem ich umieszczania.”*

  Edsger W. Dijkstra

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Kłopot z programistami jest taki, że nigdy nie wiadomo, co robią, do czasu, gdy jest za późno.”*

  Seymour Roger Cray

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Każdy głupiec może napisać kod zrozumiały dla komputera. Dobrzy programiści potrafią stworzyć kod zrozumiały dla ludzi.”*

  Martin Fowler

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Zawsze pisz kod tak, jakby gość, który ma się nim zajmować, był agresywnym psychopatą, który wie, gdzie mieszkasz.”*

  Martin Golding

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Czasami bardziej opłaca się zostać w łóżku w poniedziałek, niż spędzić resztę tygodnia, debugując kod z poniedziałku.”*

  Christopher Thompson

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Jednym z moich najbardziej produktywnych dni był ten, w którym wyrzuciłem 1000 linijek kodu.”*

  Ken Thompson

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *„Napisanie pierwszych 90% kodu aplikacji zajmuje 90% czasu pracy. Napisanie pozostałych 10% kodu zajmuje pozostałe 90% czasu pracy.”*

  Tom Cargill

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- Testowanie umożliwia bezpieczny refactoring i zmniejsza ilość debugowania i błędów.
- Brak testowania wymusza stosowanie zasady "Działa? Nie ruszaj!"
- Jak wygląda debugowanie bez testowania, obrazuje ten felieton (wiem, że niby nie na temat, ale tak mi się podoba, że nie mogłem się powstrzymać ;-)) :

Dlaczego potrzebujemy testowania i refaktoringu? cd.

- *"Kupiłem sobie klej: syndetikon w tubce. Poradzono mi, abym szyjkę tubki przekłuł szpilką, co też zrobiłem. W ten sposób z przodu powstała dziurka, przez którą za pociśnięciem wychodzić miał klej. Niestety dziurka była zbyt mała i klej wychodził ledwo ledwo. Zirytowany, pocisnąłem mocniej, ale wtedy tubka pękła i z boku zrobiła się duża dziura, przez którą wylazło sporo kleju, plamiąc mi palce. W ten sposób były już dwie dziury, przy czym oczywiście więcej wychodziło przez duże pęknięcie niż przez maleńką dziurkę legalną. Wobec tego postanowiłem poczekać, aż zewnętrzna, na powietrzu się znajdująca warstwa kleju w dużej dziurze zastygnie, tworząc w ten sposób korek naturalny, a tymczasem powiększyć dziurkę z przodu, wiercąc w niej szpilką. Tak też zrobiłem, przy wierceniu jednak szpilką w twardym i gęstym kleju zdarzyła się mała katastrofa: szpilka skrzywiła się i wykłuła w drugim boku tubki jeszcze jedną dziurkę. W ten sposób dziur było już obecnie trzy. Badałem je, przyciskając koniec tubki palcami, i stwierdziłem, że hierarchia wychodzenia kleju była teraz następująca: najwięcej wyłaziło przez owo duże pęknięcie, później przez dziurkę zrobioną ostatnio szpilką, najmniej przez legalny otwór w szyjce. W dodatku jakkolwiek by się tubkę trzymało, musiało się mieć poplamione palce, a w ogóle połowa zawartości tubki wylazła mi już do ręki, sklejając kciuk ze środkiem dłoni, tak że postanowiłem wyskrobać to miejsce nożyczkami od "manikiuru", co mi, lewą ręką, poszło bardzo niesporo; w rezultacie skaleczyłem się dotkliwie i musiałem odłożyć tubkę na dziesięć minut, w czasie których na przemian jodynowałem rękę i obmywałem ją z kleju. Powróciwszy do tubki stwierdziłem, że wszystkie trzy dziurki zastygły teraz na kamień, wobec czego nastąpiło zrównanie szans między nimi i można było rozpocząć całą akcję od początku. Postanowiłem jeszcze raz przebić szyjkę szpilką, lecz w porę przypomniałem sobie, że znowu może się to skończyć jej (szpilki) skrzywieniem i wybiciem przypadkowej, niepożądanej dziury. Wobec tego postanowiłem obciąć narosły u wylotu szyjki skamieniały klej scyzorykiem. Tak też zrobiłem, lecz mimo to za pociskaniem tubki nic nie wychodziło, widocznie wewnątrz szyjki utworzył się jakiś skrzep ze stwardniałego kleju, skrzep sięgający daleko w głąb. Wobec tego zdecydowałem się obciąć scyzorykiem całą szyjkę. Była ona metalowa, toteż długo i mozolnie ją piłowałem, gdy nagle znów nastąpiła katastrofa: najwidoczniej za mocno przyciskałem szyjkę scyzorykiem, wytworzyło się parcie i oto wszystkie blizny po poprzednich dziurach otworzyły się, a prócz tego klej znalazł sobie jeszcze jedno, tym razem generalne ujście, bo cały tył tubki odwinął się, tworząc szeroki kanał, którym reszta zawartości wytoczyła mi się do ręki i na ubranie. W ten sposób wszystko się skończyło: niepotrzebną już tubkę, a raczej to, co nią kiedyś było, wyrzuciłem do kubła ze śmieciami (śmieci zresztą natychmiast się posklejały), umyłem ręce, oczyściłem ubranie i, zmęczony nieco oraz zbaraniały, usiadłem przy oknie."*

  Stefan Kisielewski, 1952

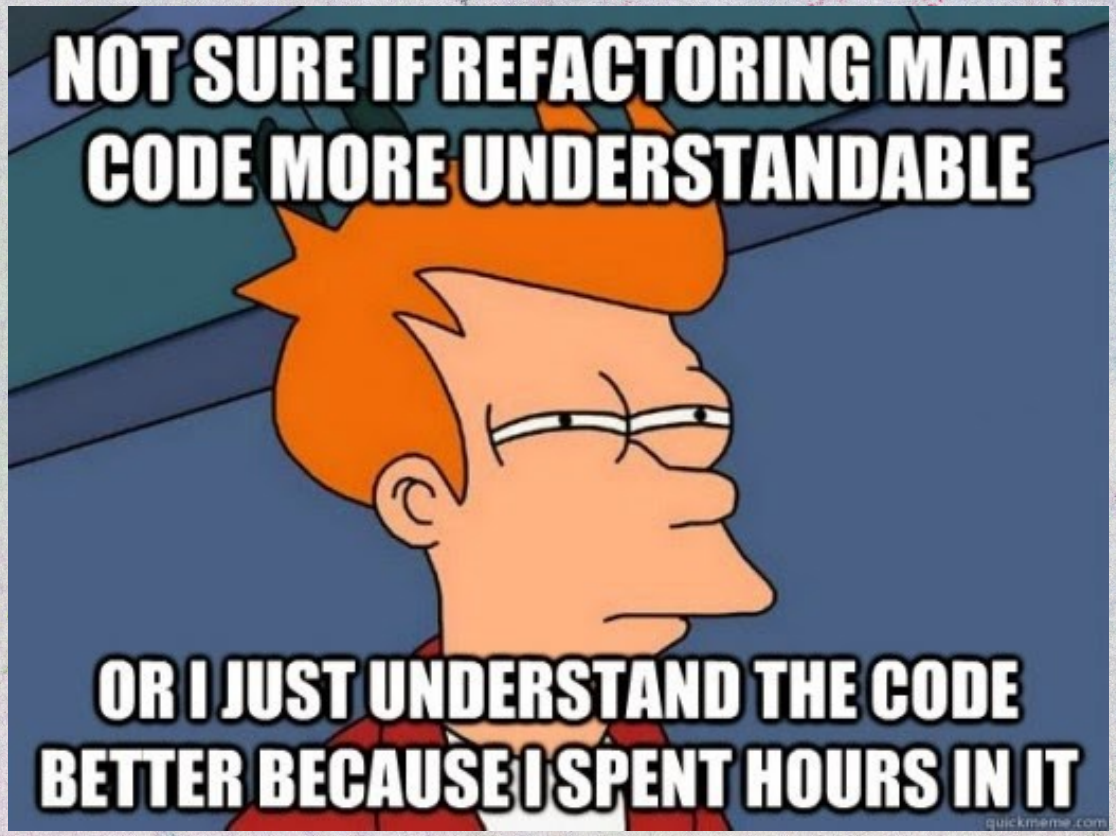
Antywzorce

- **S**ingleton
- **T**ight Coupling
- **U**ntestability
- **P**remature Optimization
- **I**ndescriptive Naming
- **D**uplication

Wzorce

- **S**ingle Responsibility Principle
- **O**pen/Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

Krytyka / kontrowersje

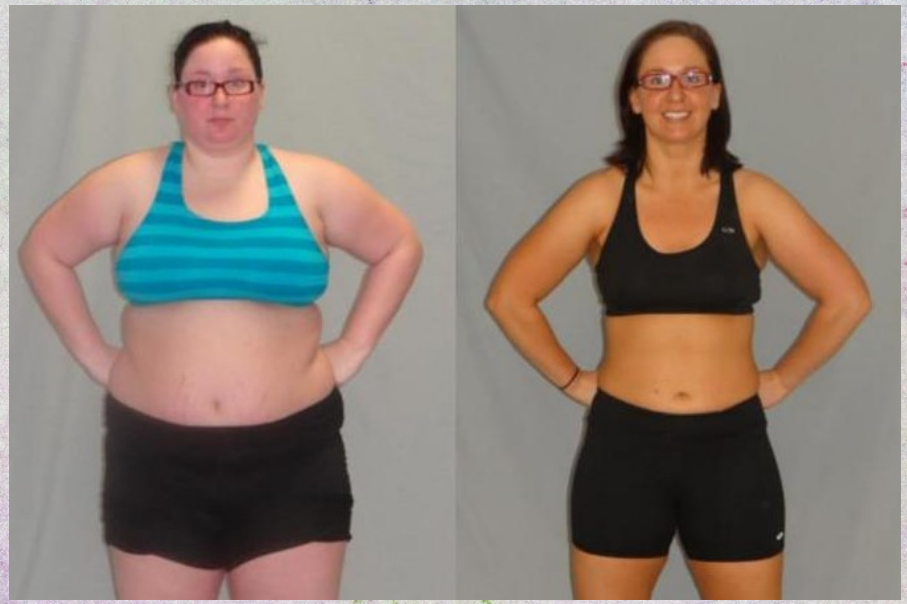


Krytyka / kontrowersje

- Sama faktoryzacja może wytworzyć mnóstwo dodatkowych błędów.

Krytyka / kontrowersje

Nie ma nic gorszego, niż złe metody w dobrej sprawie. Na pewno znacie taki typ reklam („przed” i „po”) :



Na zdjęciu po prawej kobieta ma:

- 1) lepiej wyglądajacy biustonosz,
- 2) kolory zdjęcia są lepsze (nasycenie / paleta),
- 3) lepiej dobraną fryzurę,
- 4) słabiej widoczne okulary,
- 5) ale przede wszystkim **UŚMIECHA SIĘ!** A jest udowodnione wieloma badaniami socjologicznymi, że osoby, które się uśmiechają, są bardziej lubiane / popularniejsze, jednym słowem robią lepsze wrażenie.

Możliwe jest więc, że gdyby na zdjęciu z lewej strony kobieta się uśmiechała, a na zdjęciu z prawej nie, to wrażenie byłoby mimo wszystko przeciwne...

Krytyka / kontrowersje

Przeglądając literaturę nt. prefactoringu i wzorców projektowych zauważyłem, że niemal wszystkie przykłady mające ilustrować przewagę zalecanych metod są tego typu, co zestawienie zdjęć na poprzedniej stronie, tzn. są tendencyjne, porównują więcej niż jedną rzecz, przy czym autorzy udają, że porównują tylko jedną; są więc jak wcześniej pokazana reklama. Sprawia to niestety wrażenie, że teza nie jest słuszna, bo tylko kłamstwo potrzebuje manipulacji.

Krytyka / kontrowersje - przykłady

- Przykład z „Pragmatycznego programisty” Andrew Hunta i David Thomasa. Porównanie niby tych samych czynności różnymi sposobami. W Shellu ma być lepsze, bo krótsze::
- **Shell:**

  find . -name '.c' -newer Makefile -print*
- **GUI:**

  *Open the Explorer, navigate to the correct directory, click on the Makefile, and note the modification time. Then bring up Tools/Find, and enter *.c for the file specification. Select the date tab, and enter the date you noted for the Makefile in the first date field. Then hit OK.*

- Do powyższego mam kilka uwag:

  1) Nie znam nikogo, kto pisałby notatki w ten sposób, jak tu są napisane przy GUI. Jeśli już, to ew. notuje się np. tak:

  *Explorer, find directory, note Makefile date , ctrl-F -> *.c, date -> first field*

  OK pomijamy jako oczywiste. I co? No i niby dalej przykład z GUI jest dłuższy, ale już to porównanie nie robi takiego, hm, *piorunującego* wrażenia

  2) W przykładzie z shellem **zakłada się, że jesteśmy we właściwym katalogu, a w GUI mamy go wyszukać (sic!).** Po poprawce wersja GUI wygląda tak: *note Makefile date , ctrl-F -> *.c, date -> first field* . Teraz obie wersje wyglądaja na podobnej długości.

  3) W przykładzie z GUI **zakłada się,** że okno z shellem jest już otwarte. Dlaczego? Nie wiadomo. Równie dobrze można np. założyć, że okno Explorera jest otwarte, a shella nie. Wersja Shellowa zaczyna wyglądać na dłuższą.
- 4) W przykładzie shellowym skopiowany jest tylko i wyłącznie tekst, który wpisujemy. Gdyby analogicznie postąpić tak z przykładem z GUI, wyglądałby on np. tak: **.c* , *04/28/18* . Zgadza się, tylko tyle, bo tylko to wpisujemy. Tendencyjne? Oczywiście. Tak jak podany w książce przykład...

  5) W wersji GUI opisane są **czynności**, które wykonujemy, a w przykładzie z shellem wyłącznie efekty tych czynności. Skoro w GUI opisane jest, co wciskamy, to analocznie powinien być zapisany przykład z lewej, czyli np.

Krytyka / kontrowersje – przykłady i kontrprzykłady

Check out if the light “Caps Lock” on the computer keyboard is off. If it is on, press the key Caps Lock on the computer keyboard. Find and press a key on the computer keyboard with a letter “F” on it. Realize it. Find and press a key on the computer keyboard with a letter “I” on it. Realize it. Find and press a key on the computer keyboard with a letter “N” on it. Realize it. Find and press a key on the computer keyboard with a letter “D” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with “.” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with “-” on it. Realize it. Find and press a key on the computer keyboard with a letter “N” on it. Realize it. Find and press a key on the computer keyboard with a letter “A” on it. Realize it. Find and press a key on the computer keyboard with a letter “M” on it. Realize it. Find and press a key on the computer keyboard with a letter “E” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with ' on it. Realize it. Find and press a key on the computer keyboard with “” on it. Realize it. Find and press a key on the computer keyboard with “.” on it. Realize it. Find and press a key on the computer keyboard with a letter ”C” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with “-” on it. Realize it. Find and press a key on the computer keyboard with a letter “N” on it. Realize it. Find and press a key on the computer keyboard with a letter “E” on it. Realize it. Find and press a key on the computer keyboard with a letter “W” on it. Realize it. Find and press a key on the computer keyboard with a letter “E” on it. Realize it. Find and press a key on the computer keyboard with a letter “R” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with letters “Shift” on it. Find and press a key on the computer keyboard with a letter “M” on it. Realize it and the key with letters „Shift”. Find and press a key on the computer keyboard with a letter “A” on it. Realize it. Find and press a key on the computer keyboard with a letter “K” on it. Realize it. Find and press a key on the computer keyboard with a letter “E” on it. Realize it. Find and press a key on the computer keyboard with a letter “F” on it. Realize it and the key with letters „I” on it. Realize it. Find and press a key on the computer keyboard with a letter “L” on it. Realize it. Find and press a key on the computer keyboard with a letter “E” on it. Realize it. Find and press a key on the computer keyboard with nothing on it. Realize it. Find and press a key on the computer keyboard with a “-” on it. Realize it. Find and press a key on the computer keyboard with a letter “P” on it. Realize it. Find and press a key on the computer keyboard with a letter “R” on it. Realize it. Find and press a key on the computer keyboard with a letter “I” on it. Realize it and the key with letters „N” on it. Realize it. Find and press a key on the computer keyboard with a letter “T” on it. Realize it. Find and press a key on the computer keyboard with letter “Enter” on it. Realize it*

- Porównajcie teraz z wersją GUI:

  *note Makefile date , ctrl-F -> *.c, date -> first field* .

Krytyka / kontrowersje - przykłady

- Przykład z „Czystego kodu”

```
public List<int[]> getThem() {
  List<int[]> list1 = new ArrayList<int[]>();
  for (int[] x : theList)
    if (x[0] == 4)
      list1.add(x);
  return list1;
}
```

- I zrefakturowany:

```
public List<Cell> getFlaggedCells() {
  List<Cell> flaggedCells = new ArrayList<Cell>();
  for (Cell cell : gameBoard)
    if (cell.isFlagged())
      flaggedCells.add(cell);
  return flaggedCells;
```

Dlaczego te przykłady są tendencyjne? Brak jest w pierwszym przykładzie… komentarzy. W starym styu zawsze się podkreślało, że należy dobrze komentować niejasne fragmenty kodu, Oczywiście możliwe, że nadal drugi przykład wyglądałby lepiej, ale nie przekonamy się o tym, bo komentarze zostały wycięte... Porównanie kiepskiego fragmentu starego stylu z dobrym fragmentem nowego wcale nie udowadnia wyższości tego drugiego.

Wnioski

- Jakie stąd płyną wnioski? Dobrze by było porównać stare metody programowania z metodami czystego kodu w bardziej bestronny sposób, najlepiej przez jakąś grupę bestronnych osób. Tyle że… takie coś już się odbyło, jak przeczytałem na https://sourcemaking.com/ :

  The study of design patterns has been excessively ad hoc, and some have argued that the concept sorely needs to be put on a more formal footing. At OOPSLA 1999, the Gang of Four were (with their full cooperation) subjected to a show trial, in which they were "charged" with numerous crimes against computer science. They were "convicted" by ⅔ of the "jurors" who attended the trial.

Krytyka - ciekawostki

- Również na stronie sourcemaking.com znalazłem link do artykułu “Revenge of the Nerds” Paula Grahama, krytycznie nastawionego do wzorców projektowych i refactoringu, http://www.paulgraham.com/icad.html

  Niestety czytając ten artykuł łatwo zauważyć, że autor jest nie tyle negatywnie nastawiony do refactoringu, ile w ogóle do programowania obiektowego, a ideałem jest dla niego Lisp, do którego ponoć stopniowo dążą inne języki. Popularność Programowania Obiektowego ma być spowodowana tym, że większość programistów nie jest wybitna, tylko “mediocre”. No ale są to fakty, kto nie akceptuje faktów, wystawia złą ocenę własnej inteligencji...

Uwagi osobiste

- C++ jest bardzo uniwersalnym językiem, nadaje się do wielu rzeczy. Wzorce projektowe powodują ograniczenie jego możliwości oraz standaryzują kod. Mam wrażenie, że w rezultacie refaktoringu otrzymujemy kod, który jest swego rodzaju językiem wyższego poziomu niż samo C++, z tym, że programista wykonuje pracę prekompilatora tego języka. Nie wiem, co to za język, być może jest to coś w rodzaju PL/SQLa, bo udział algorytmów w tak zaprojektowanym kodzie jest stosunkowo mały, za to jest mnóstwo zależności między strukturami. Jak dla mnie jest to fascynujące, jednak nie do końca jestem przekonany, czy aby na pewno taki kod jest czytelniejszy od kodu starego typu, przede wszystkim z powodu ogromnego rozczłonkowania, wątpliwości pogłębiają też tendencyjne przykłady i niekonsekwentna argumentacja. Niepokojąca jest też możliwość wystąpienia błędów w wyniku samego refaktoringu, chociaż być może pozwala on zapobiec przyszłym błędom i pomaga w znalezieniu już istniejących. Myślę też, że mnóstwo rzeczy w ramach refactoringu można poddać automatyzacji pełnej lub częściowej i pewnie w tym kierunku będzie szedł rozwój narzędzi programistycznych, zresztą już jest dostępnych sporo narzędzi, niestety głównie do Visuala.

Literatura

- http://www.amazon.co.uk/Clean-Code-Handbook-Software-Craftsmanship/dp/0132350882/
- http://www.amazon.com/Pragmatic-Programmer-Journeyman-Master/dp/020161622X/
- http://www.amazon.com/Code-Complete-Practical-Handbook-Construction/dp/0735619670/
- http://www.amazon.com/The-Mythical-Man-Month-Engineering-Anniversary/dp/0201835959/

Narzędzia

dla Visual Studio:

- JustCode
- ReSharper
- Coderush
- Visual Assist
- Refactor! - free
- Whole Tomato Software‘s Visual Assist
- SharpSort

inne:

- DMS Software Reengineering Toolkit (Implements large-scale refactoring for C, C++, C#, COBOL, Java, PHP and other languages)
- Sigasi Studio - standalone or plugin software for VHDL and System Verilog
- XCode
- SlickEdit
- Eclipse CDT (C/C++ Development Tooling)