Jak pisać nieutrzymywalny kod

czyli jak być niezastąpionym w pracy

Szymon Gut

15 sierpnia 2012

1 Utrzymywanie kodu

2 Projekt i struktura programu

3 Nazewnictwo

4 Dokumentacja

5 Testowanie

Inżynieria oprogramowania

Inżynieria oprogramowania

- duże, długie projekty, rotacja ludzi

Inżynieria oprogramowania

- duże, długie projekty, rotacja ludzi
- konieczność czytania i modyfikowania cudzego kodu

Inżynieria oprogramowania

- duże, długie projekty, rotacja ludzi
- konieczność czytania i modyfikowania cudzego kodu
- często: potrzebny prosty bugfix/feature

Inżynieria oprogramowania

- duże, długie projekty, rotacja ludzi
- konieczność czytania i modyfikowania cudzego kodu
- często: potrzebny prosty bugfix/feature
- nie ma czasu czytać całego kodu, najlepiej lokalne zrozumienie i drobna zmiana

Cel

Cel

- maksymalne utrudnienie znalezienia właściwego miejsca w kodzie

Cel

- maksymalne utrudnienie znalezienia właściwego miejsca w kodzie
- uniemożliwienie zignorowania czegokolwiek

Cel

- maksymalne utrudnienie znalezienia właściwego miejsca w kodzie
- uniemożliwienie zignorowania czegokolwiek
- zwielokrotnienie liczby potrzebnych modyfikacji

Cel

- maksymalne utrudnienie znalezienia właściwego miejsca w kodzie
- uniemożliwienie zignorowania czegokolwiek
- zwielokrotnienie liczby potrzebnych modyfikacji

Uwaga!
Kod nie powinien **wyglądać** na nieutrzymywalny,
powinien **być** nieutrzymywalny.

Zależności

~~Don't Repeat Yourself~~

Zależności

~~Don't Repeat Yourself~~ Write Everything Twice

Zależności

~~Don't Repeat Yourself~~ Write Everything Twice

- zmienne globalne oraz statyczne

Zależności

~~Don't Repeat Yourself~~ Write Everything Twice

- zmienne globalne oraz statyczne
- zmienne ustawiane wszędzie, ale nie czytane nigdzie

Wydajność

- nie używaj enkapsulacji

Wydajność

- nie używaj enkapsulacji
- copy-paste programming

Wydajność

- nie używaj enkapsulacji
- copy-paste programming
- statyczne metody

Nazewnictwo

Idealnie: nazwy zmiennych nie powinny mieć nic wspólnego z zawartością.

Nazewnictwo

Idealnie: nazwy zmiennych nie powinny mieć nic wspólnego z zawartością.

Przykłady:

```
Fred, asdf, aoeu, nygga
```

```
plus = prawyNawias * ( minus + siedem );
```

Jednoliterowe nazwy zmiennych

Należy stosować.

Praktycznie uniemożliwiają wyszukiwanie zmiennych przez Ctrl+F.

Jednoliterowe nazwy zmiennych

Należy stosować.

Praktycznie uniemożliwiają wyszukiwanie zmiennych przez Ctrl+F.

Uwaga!
zmienne i,j,k nie muszą być stosowane wyłącznie do licznika pętli.

```
vector<int> i[100];
double j;
```

#define

#define

- synonimy

#define

- synonimy
- `#define very_fast_copy(src, dst, size) /*nothing*/`

#define

- synonimy
- `#define very_fast_copy(src, dst, size) /*nothing*/`
- kod działający tylko w trybie `DEBUG`

Kreatywne literówki

Jeśli jesteś zmuszony używać opisowych nazw, dokonuj subtelnych literówek.

```
procedures, procesures
center, centre
AbstractParserFactory, AbstrcatParserFactory
```

Kreatywna kapitalizacja

```
String filename;
String fileName;

ComputeHistogram();
ComputeHistoGram();
```

Abstrakcja

Używaj abstrakcyjnych określeń.

Abstrakcja

Używaj abstrakcyjnych określeń.

przykłady nazw zmiennych:
`data, handle, stuff, number, everything`

Abstrakcja

Używaj abstrakcyjnych określeń.

przykłady nazw zmiennych:
`data, handle, stuff, number, everything`

przykłady nazw funkcji:
`routine(), DoIt(), perform_computations(), executeMethod()`

Recykling

Recykling

- przesłanianie nazw globalnych lokalnymi

Recykling

- przesłanianie nazw globalnych lokalnymi
- przeładowywanie nazw funkcji robiących zupełnie co innego

Recykling

- przesłanianie nazw globalnych lokalnymi
- przeładowywanie nazw funkcji robiących zupełnie co innego
- zmienne tymczasowe przechowujące wiele różnych rzeczy

Komentowanie kodu

Komentowanie kodu

Kod należy komentować.

```
++i;          // add 1 to i
z   = f(x);   // call f
```

Komentowanie kodu

Kod należy komentować.

```
++i;          // add 1 to i
z   = f(x);   // call f
```

Komentarze powinny mówić, co kod robi,
a nie dlaczego i po co.

Zgodność z kodem

Nie musisz kłamać w komentarzach.

Zgodność z kodem

Nie musisz kłamać w komentarzach.

Wystarczy nie utrzymywać komentarzy aktualnych względem kodu.

Testy jednostkowe

Testy jednostkowe

Nie pisz testów. Testy są dla tchórzy.

Testy jednostkowe

Nie pisz testów. Testy są dla tchórzy.

Jeśli musisz, postaraj się, żeby niektóre nie przechodziły raz na 100 uruchomień.

Koniec

Życzę trwałego zatrudnienia.

na podstawie:
„How To Write Unmaintainable Code”, Roedy Green
`http://thc.org/root/phun/unmaintain.html`