Functii, pointeri la functii, good vs. bad code

transmiterea parametrilor către funcții

- transmitere prin valoare (pass-by-value)
  - se face o copie a parametrului și acesta este folosit de funcție
  - modificările aduse copiei se pierd după terminarea execuției funcției
- transmitere prin referință (pass-by-reference)
  - se transmite o referință către parametrul real
  - modificările aduse parametrului se păstrează și după terminarea execuției funcției

transmitere prin referință

- http://stackoverflow.com/questions/373419/whats-the-difference-between-passing-by-reference-vs-passing-by-value#answer-430958

  - If I tell you the URL, I'm **passing by reference.** You can use that URL to see the **same web page** I can see. If that page is changed, we both see the changes. If you delete the URL, all you're doing is destroying your reference to that page - you're not deleting the actual page itself.
  - If I print out the page and give you the printout, I'm **passing by value**. Your page is a disconnected copy of the original. You won't see any subsequent changes, and any changes that you make (e.g. scribbling on your printout) will not show up on the original page. If you destroy the printout, you have actually destroyed **your copy** of the object - but the original web page remains intact.

transmitere prin valoare

- In C, toti parametrii se transmit prin valoare!

Cum se transmit parametrii în C?

- parametrii în C se transmit exclusiv prin valoare
- orice parametru trimitem unei funcții în C acestuia i se va crea o copie și în funcție va fi folosită doar copia.

- Atat tipurile primitive
- Cat si pointerii
- Cat si vectorii (array)
- Cat si tipuri compuse (struct, union)

- si atunci ce tip de transmitere avem aici?

```
int f (int *x)
{
        *x=1;
}
main()
{
        int a;
        f(&a);
}
```

- R: transmitere prin valoare

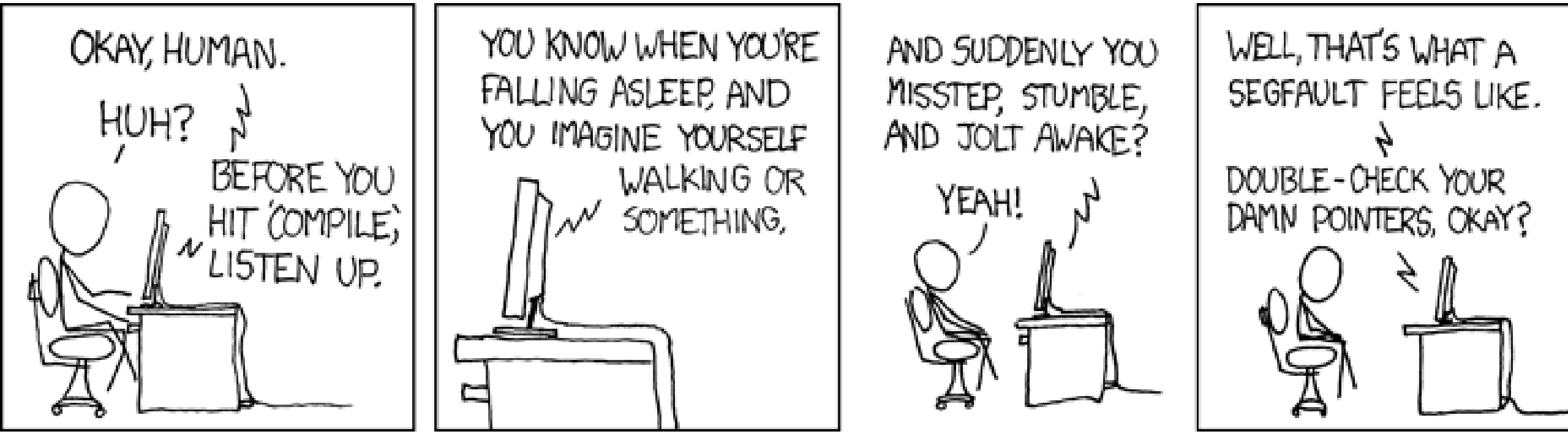
```
int f (int *x)
{
        *x=1;
}
main()
{
        int a;
        f(&a); //functiei f i se transmite o copie a adresei lui a
//prin intermediul adresei se poate accesa zona de memorie unde este pastrata valoarea lui a
}
```

transmitere prin adresă

- transmiterea prin adresă (pass by pointer) = implementarea în C a transmiterii parametrilor prin referință din C++
- se transmite funcției adresa de memorie a variabilei pe care ne dorim s-o modificăm
- funcția face o copie a adresei dar prin intermediul adresei lucrează cu variabila ”reală” (si cu zona de memorie “reală”)

problemele transmiterii prin adresă

- se pot trimite pointeri neinițializați
- se poate trimite NULL (trebuie să verificăm în funcție dacă nu am primit NULL)



problemele transmiterii prin valoare

- copierea parametrilor poate dura mult dacă transmitem tipuri de date de mari dimensiuni (structuri cu multe câmpuri de exemplu)

tipul referință (doar în C++)

- tipul referință = un pointer sigur
- int x=5;
- int& y=x;
- referințele se inițializează obligatoriu la declarare
  - mai puțin când sunt declarate cu ajutorul **extern**
  - sau când sunt parametri ai funcțiilor
- pot fi folosite ca parametri (unul din foarte puținele cazuri când nu trebuie inițializate)

avantaje referințe

- nu pot fi modificate după declarare (nu pot fi asignate să puncteze către alte variabile)
- sunt inițializate obligatoriu
- nu pot avea valoarea NULL (poate fi un dezavantaj)
- dezavantajele:
  - cam toate în afară de inițializarea obligatorie ☺
  - nu sunt suportate în C-ul standard
  - nu poți folosi aritmetica pointerilor pe variabile de tip referință
- referințe vs. pointeri
  - siguranță mai mare, flexibilitate minimă vs. flexibilitate maximă, posibilități mult mai mari de a greși

```
int main()
{
      int x=5;
       int &a=x;
       a++;
       printf("%d %d\n",a,x);
       return 0;

}
```

compilați cu g++ !

comparație referințe vs. pointeri

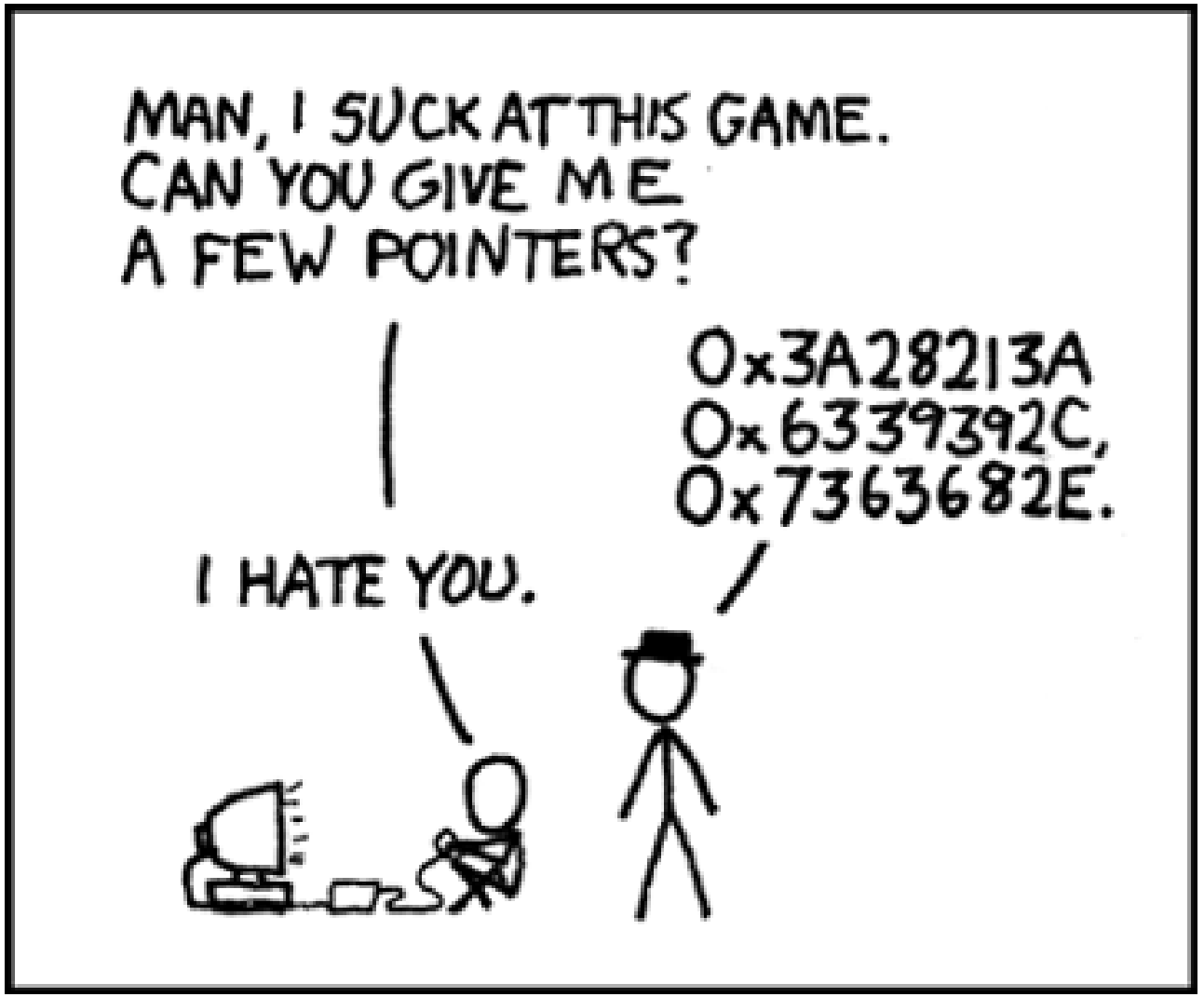
```
void swap(int &a, int &b)
{
        int temp=b;
        b=a;
        a=temp;
}
int main()
{
        int x=5, y=7;
        swap(x,y);
        printf("%d %d\n",x,y);
        return 0;

}
```

```
void swap(int *a, int*b)
{
        int temp=*b;
        *b=*a;
        *a=temp;
}
int main()
{
        int x=5,y=7;
        swap(&x,&y);
        printf("%d %d\n",x,y);
        return 0;
}
~
```

pointeri la funcții

- declarare
- utilizare
- exemple



reminder – declarare pointeri la funcții

- void (*f0)(int);
- int* (*f1)(int*, int*);
- int (*f2)(int* (*f1)(int*, int*), int*, int*);
- int (*f3[10])(int* (*f1)(int*, int*), int*, int*);
- int (**f4)(int* (*f1)(int*, int*), int*, int*);
- void (*signal(int sig, void (*func)(int)))(int);

reminder – declarare pointeri la funcții

- void (*f0)(int); //f0=pointer către o funcție care are un parametru întreg și care nu întoarce rezultat
- int* (*f1)(int*, int*); //f1= pointer către o funcție care întoarce un element de tip int* și care are 2 parametri de tip int*
- int (*f2)(int* (*f1)(int*, int*), int*, int*); //f2=pointer către o funcție care întoarce un element de tip int și care are un parametru de tip funcție (v. f1) și alți 2 parametri de tip int*
- int (*f3[10])(int* (*f1)(int*, int*), int*, int*);//f3 este un vector de 10 pointeri catre functii similare cu f2
- int (**f4)(int* (*f1)(int*, int*), int*, int*);//f4 este un pointer către un pointer la o funcție f2.
  - putem aloca dinamic un vector de pointeri la funcții)
  - sau cum putem pune 8 steluțe într-o singură declarație ☺

declarare folosind typedef

- typedef void (*handler) (int num);
- definește un tip pe care îl numim *handler*
  - pointer la funcție care primește un int și nu întoarce nimic
- sintaxa: typedef declarare_pointer. Numele tipului este dat de numele pointerului

- void (*signal (int sig, **void (*func)(int)**))(int);
- void (*func) (int) – tocmai l-am definit ca fiind tipul **handler** (typedef void (*handler) (int num);)
- **void (*signal** (int sig, handler func)) **(int);**
- **handler** signal(int sig, **handler** func);

avem o funcție care primește ca parametri un întreg și o funcție și întoarce un rezultat de tip pointer la o funcție care primește un întreg nu întoarce nimic☺

- funcția signal o veți folosi la SO (sisteme de operare) în anul 3

- http://stackoverflow.com/questions/1591361/understanding-typedefs-for-function-pointers-in-c-examples-hints-and-tips-ple

The ``Clockwise/Spiral Rule''

- http://c-faq.com/decl/spiral.anderson.html
- Plecati de la elementul necunoscut si miscati-va in spirala sensul acelor de ceas; cand intalniti unul din elementele urmatoare inlocuiti-le cu afirmatiile corespunzatoare
- [X] or []
  - => vector de dimensiune X... sau vector de dimensiune nedefinita
- (type1, type2)
  - => functie cu parametrii de tip type1 si type2 ce intoarce...
- *
  - => pointer la..
- rezolvam intotdeauna mai intai parantezele

- void (*signal (int sig, **void (*func)(int)**))(int);

utilizare pointeri la funcții

- Scrieţi un program care afişează valorile functiilor sqrt, sin, cos, tan, exp şi log, în intervalul [1..10], cu pasul 0.1. În acest scop, se creează un tablou de pointeri la aceste funcţii şi se apelează funcţiile în mod indirect prin aceşti pointeri.

```c
#include<math.h>
#include<stdio.h>
typedef double (*fp) (double);
int main()
{
        float i; int j;
        fp farr[6]={sin, cos, log,exp, tan, sqrt};
        for(i=0;i<10;i+=0.1)
        {
                for(j=0;j<6;j++)
                        printf("farr[%d](%f)=%lf ",j,i,farr[j](i));
                printf("\n");

        }
        return 0;
}
```

```c
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <signal.h>

void sighandler(int);

int main()
{
   signal(SIGINT, sighandler);

   while(1)
   {
      printf("Going to sleep for a second...\n");
      sleep(1);
   }

   return(0);
}

void sighandler(int signum)
{
   printf("Caught signal %d, coming out...\n", signum);
   exit(1);
}
```

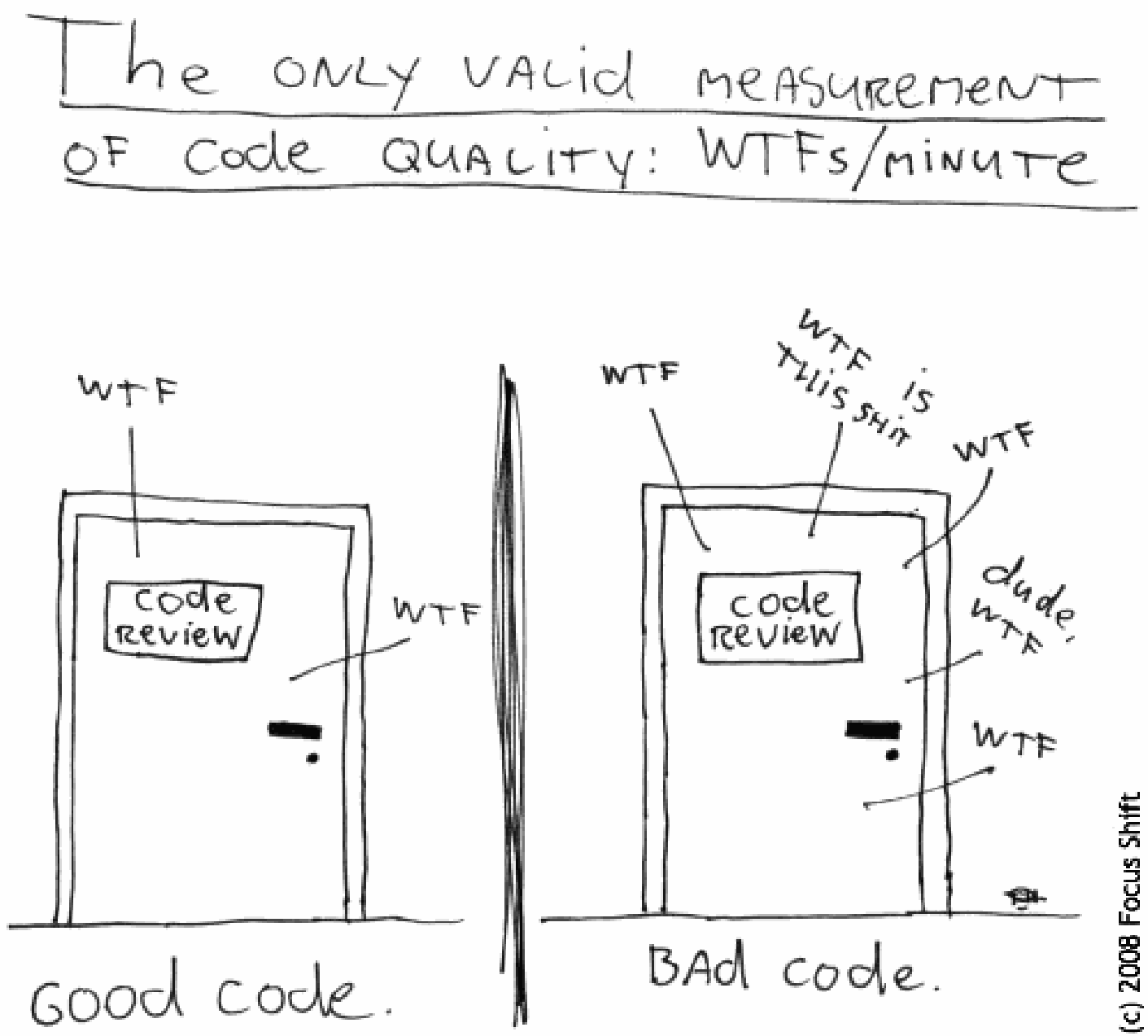
Try it

http://www.tutorialspoint.com/c_standard_library/c_function_signal.htm

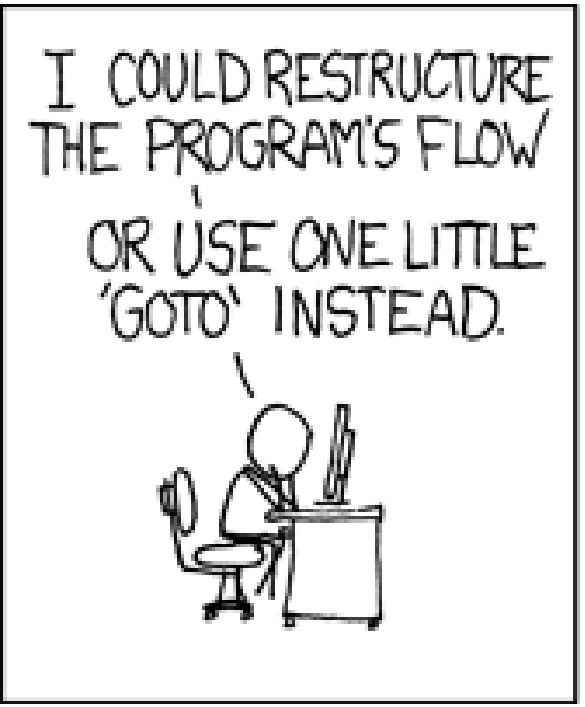
Cum scriem ”good code”?

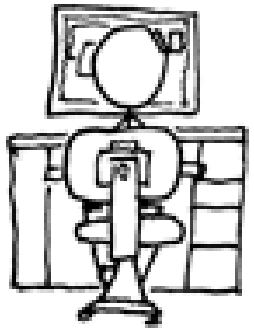
- code review
- best practices
- gândim și apoi codăm
- comentarii
- KIS(S)

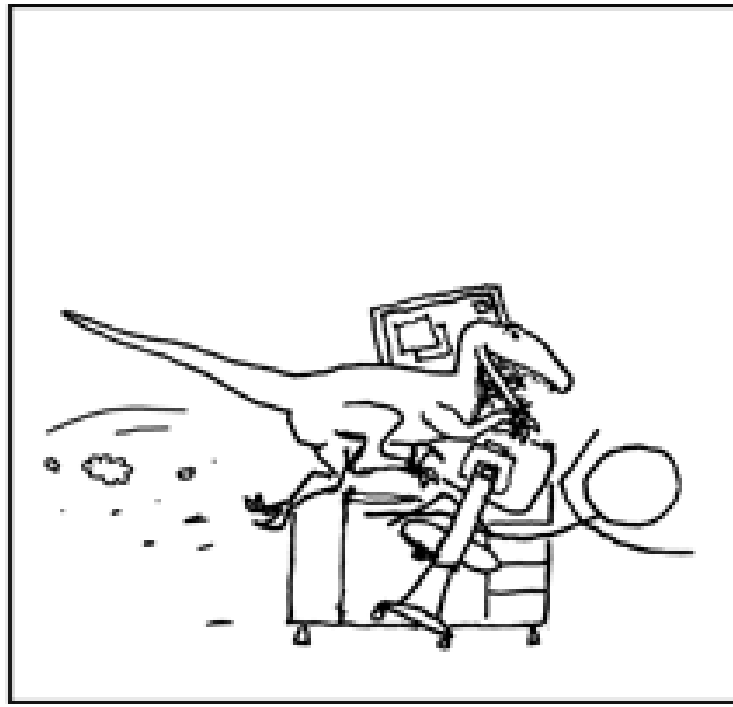
code review


The ONLY VALID MEASUREMENT OF Code QUALITY: WTFs/MINUTE
WTF
code review
WTF
Good code.
WTF
WTF is THIS SHIT
WTF
code review
dude, WTF
WTF
BAd code.
(c) 2008 Focus Shift

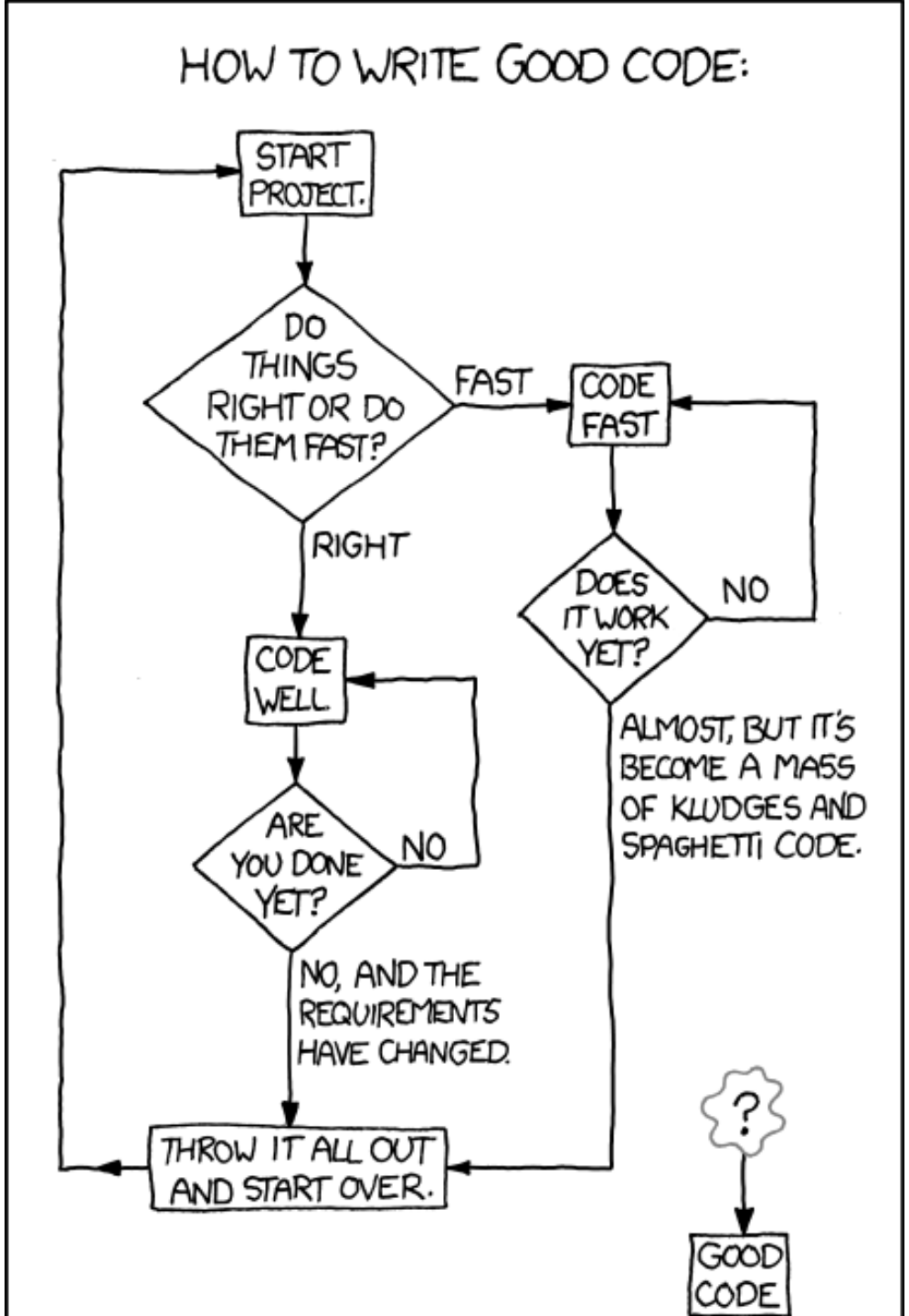
best practices vs. ”dar merge și așa”



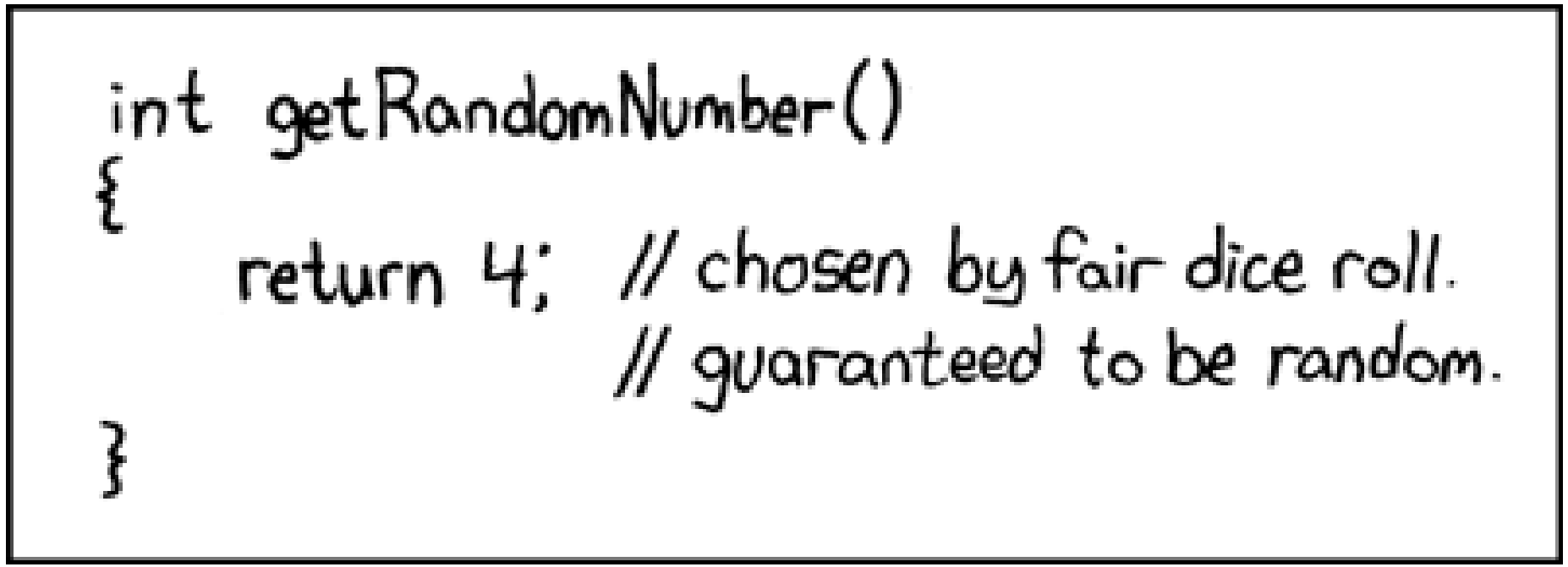




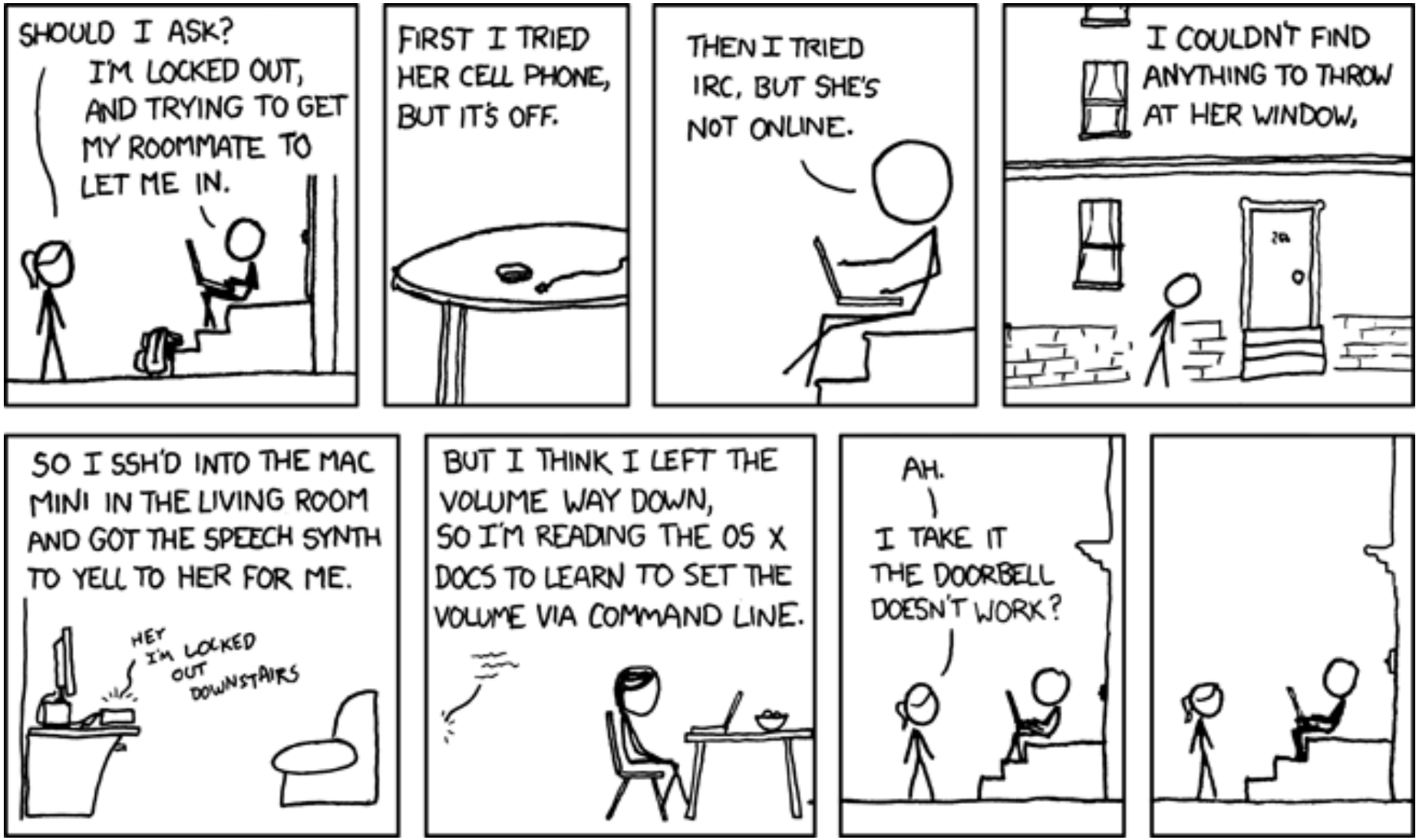
gândim și apoi codăm



comentarii relevante



Keep **I**t **S**imple (dacă se poate)



Referințe

- referințele către poze lipsesc de dragul productivității voastre. Le adaug în vacanță ☺